Two-dimensional Stroke Gesture Recognition: a Survey

NATHAN MAGROFUOCO, Université catholique de Louvain, Belgium
PAOLO ROSELLI, Università di Roma "Tor Vergata", Italy and Univ. cath. de Louvain, Belgium
JEAN VANDERDONCKT, Université catholique de Louvain, Belgium

The expansion of touch-sensitive technologies, ranging from smartwatches to wall screens, triggered a wider use of gesture-based user interfaces and encouraged researchers to invent recognizers that are fast and accurate for end-users while being simple enough for practitioners. Since the pioneering work on two-dimensional (2D) stroke gesture recognition based on feature extraction and classification, numerous approaches and techniques have been introduced to classify uni- and multi-stroke gestures, satisfying various properties of articulation-, rotation-, scale-, and translation-invariance. As the domain abounds in different recognizers, it becomes difficult for the practitioner to choose the right recognizer depending on the application and for the researcher to understand the state-of-the-art. To address these needs, a targeted literature review identified sixteen significant 2D stroke gesture recognizers that were submitted to a descriptive analysis discussing their algorithm, performance, and properties, and a comparative analysis discussing their similarities and differences. Finally, some opportunities for expanding 2D stroke gesture recognition are drawn from these analyses.

CCS Concepts: • **Human-centered computing** → **Gestural input**; **Graphical user interfaces**; **Touch screens**; • **Software and its engineering** → *Graphical user interface languages.*

Additional Key Words and Phrases: Gesture-based interfaces, Gesture recognition, Stroke gestures, Touch gestures

ACM Reference Format:
Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2021. Two-dimensional Stroke Gesture Recognition: a Survey. *ACM Comput. Surv.* 54, 7, Article 155 (July 2021), 34 pages. https://doi.org/10.1145/3465400

1 INTRODUCTION

End-users basically interact with an application or a computer system through two types of gestures [44, 90]: *contact-based* gestures which require the sampling of points by a touch-sensitive interactive surface and *contactless gestures* which do not impose this constraint, thus using computer vision techniques [30]. New touch-sensitive technologies, such as smartwatches, smartphones, tablets, notebooks, embedded devices, and wall screens, enable end users to interact with any pointing device (*e.g.*, a mouse, a pen, a stylus, a laser pointer, or any combination of them) or fingers. Gesturing has become an important interaction paradigm amongst such technologies to input text and commands on computers, a historical challenge [56] in Human-Computer Interaction (HCI) [90].

Gesture-based interaction offers an alternative to the traditional keyboard, mouse, and direct manipulation User Interfaces (UIs). Moreover, gesturing is affordable and efficient since no additional hardware resource, like screen space or hardware buttons, is required. Gesturing is also a natural way to convey information and meaning

Authors' addresses: Nathan Magrofuoco, nathan.magrofuoco@uclouvain.be, Université catholique de Louvain, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium; Paolo Roselli, Università di Roma "Tor Vergata", Vaile della Ricerca Scientifica, Roma, 00177, Italy, Univ. cath. de Louvain, Place des Sciences, 2, Louvain-la-Neuve, 1348, Belgium, roselli@mat.uniroma2.it,paolo.roselli@uclouvain.be; Jean Vanderdonckt, jean.vanderdonckt@uclouvain.be, Université catholique de Louvain, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium.

0360-0300/2021/7-ART155 $15.00
https://doi.org/10.1145/3465400

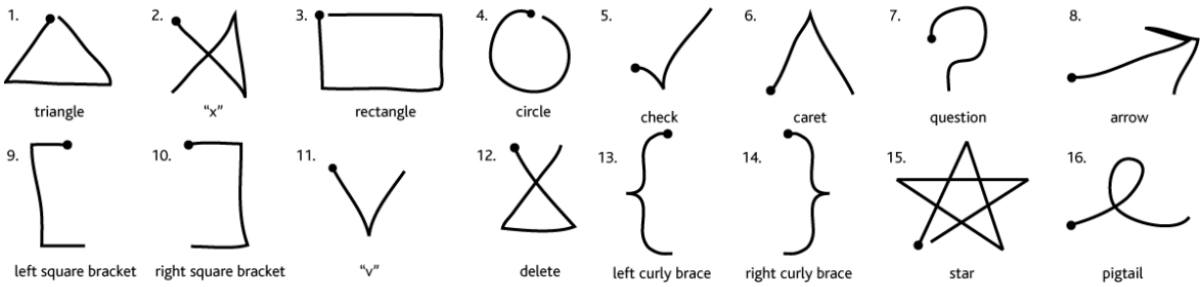


Fig. 1. The $1 dataset comprises 16 two-dimensional uni-stroke gestures, including letters, punctuation marks, geometric shapes, and symbols. The gesture for the "x" letter is imposed by the uni-stroke recognition capability of $1.

between humans [51, 79]. As gesture-based UIs become ubiquitous via the widespread use of touch-sensitive surfaces, the need for high-quality gesture recognition also continues to rise.

While several surveys are available for gesture recognition in general [29], or in particular for some limb (*e.g.*, for the hand [2, 30]), some device (*e.g.*, the Microsoft Kinect [11]), in some environment (*e.g.*, the home [42]), using some technique (*e.g.*, computer vision [52]), for other types of gestures (*e.g.*, 3D gestures [17, 38]), or for any combination of them (*e.g.*, computer vision for hand recognition [52]), there is no counterpart for 2D stroke gesture recognition.

Stroke gesture recognition refers to the classification of any movement trajectories entered with a pointing device or fingers used as shortcuts to indicate scope and commands [55, 56, 90, 91]. Two-dimensional (2D) stroke gestures are constrained by any touch-sensitive surface and typically encompass letters, digits [84], punctuation marks [86], lines, markings [24], geometric shapes, symbols [53], and other patterns [69]. For example, Fig. 1 illustrates some 2D stroke gestures belonging to the $1 dataset [85], one the most used gesture sets to evaluate the performance of uni-stroke gesture recognizers. 2D stroke gestures are usually sampled by a touch-sensitive surface onto which they are traced, and encoded as time-ordered sequences of 2D (x, y) points [40, 62, 69, 90]: $G{=}(x_i, y_i),\ \forall i \in [1, \ldots, n]$. Optionally, stroke gestures include a third, even a fourth dimension when encoded as (x, y, t, p) when a timestamp t_i or an identifier id_i, and the pressure aspect p_i are manipulated [90]. Gesture points are sampled at a rate determined by the sensing hardware and software [85, 91]. Therefore, a 2D stroke gesture recognizer is aimed at classify an unknown *candidate gesture* by comparing it against pre-recorded *template gestures* gathered by *gesture classes* in a *training set* that was used to train the recognizer beforehand. This assumes calculating a (dis-)similarity distance between the candidate gesture and template classes.

Historically, the Rotate-Scale-Translate (RST) paradigm [36] became a *de facto* standard to characterize flat stroke gestures: from two reference points captured, or more, meaningful gesture parameters, or *features*, are computed to establish *rotation* (relying on angular data), *scaling* (based on length ratio), and *translation* (linked to the barycenter of the translation). These three fundamental operations can be effectively performed on any stroke gesture. Consequently, the invariance with respect to rotation, scaling, and translation became vital properties to make any recognizer insensitive to gestures variations along these dimensions. Similarly, the order-number-direction invariance, or *articulation-invariance* for short, accommodates gesture variations in terms of how strokes are produced in time, how many (uni-stroke *vs.* multi-stroke), and in which direction [26].

This survey concerns 2D stroke gesture recognizers, which should be differentiated from other gesture interaction techniques (*e.g.*, full-body and hand gestures) and other families of classifiers (*e.g.*, handwriting and sketch recognizers). The scope is delineated based on the inclusion/exclusion criteria as follows:

(1) The gestures are performed only on 2D interactive surfaces, which are easy to sample, to process, and to recognize, also avoiding the segmentation problem of 3D gestures in thin air. The starting and the ending of the gestures are explicit both for the sensing mechanism and for the end user's proprioception: a gesture

starts as soon as a pointing device or a finger touches the surface and ends when it leaves the surface [90]. One-dimensional (1D) gestures are assimilated to 2D gestures with one dimension neglected (*e.g.*, the "swipe" refers to producing a horizontal line whose y coordinate remains more or less constant).

(2) A stroke gesture can be articulated in a single (*uni-stroke*) or in multiple strokes (*multi-stroke*), as they are often differentiated in the literature. For example, the letter "x" should be drawn in one shot when the recognizer is uni-stroke (see Fig. 1) or as two crossed lines when it is multi-stroke.
(3) As touch-sensitive surfaces allow using multiple fingers simultaneously, some differentiate uni-touch from multi-touch gestures. Multi-touch (or multi-path) recognition is excluded from this survey even though several recognizers compared hereafter have been extended to classify multi-touch gestures [8, 10, 54].
(4) Handwriting on a computer system is a "natural" and fluid mode of text entry, thanks to the user's prior experience from writing on paper. If handwriting recognition systems allow end-users to enter text either as a cursive script or as individual letters, stroke gesture recognition systems exclusively aim to classify individual letters, avoiding the character segmentation problem. In contrast, 2D stroke gestures include more gestures than letters and digits, encompassing geometric shapes and other individual symbols. We refer to [41, 49] for comprehensive surveys on on-line and off-line handwriting recognition.
(5) For the same reasons, gesture-based diagramming and sketching are considered beyond the scope of this survey. Nevertheless, handwriting, diagramming, sketching, and stroking are subsets of pattern matching recognition and several recognizers can be effective in classification and recognition [47].

As the domain abounds in different recognizers, choosing the right recognizer depending on an application and understanding the state-of-the-art become challenging. Therefore, this survey targets the following audience: for the *designer* to select the right recognizer depending on the constraints imposed by the application, for the *developer* to understand how to implement the selected recognizer, and for the *researcher* to elaborate on the existing state-of-the-art. For this purpose, section 2 conducts a targeted literature review to identify sixteen significant 2D stroke recognizers that were submitted to a systematic descriptive analysis of their motivations, the underlying algorithm, the satisfied properties of invariance, and the reported performance. Section 3 performs a comparative analysis of these recognizers to assess their similarities and differences based on their classification techniques as well as the evaluation protocols, the properties, the datasets, their modalities of input, and the (dis)similarity metrics. Section 4 suggests a decision table for choosing the right recognizers, and challenges and opportunities for future advances in the domain.

2 TWO-DIMENSIONAL STROKE GESTURE RECOGNIZERS

We conducted a Targeted Literature Review (TLR), which is a non-systematic, in-depth, and informative literature review, that keeps only the references maximizing rigorousness and relevance while minimizing selection bias [37]. The relevant references obtained may be more limited than in a Systematic Literature Review (SLR) [32], but stem from a knowledgeable selection of high-quality, easy-to-identify references on 2D stroke gesture recognition, as opposed to an all-encompassing list of somewhat relevant references. This applies to 2D gesture recognition, where the literature is abundant. Based on this TLR, we identified 16 two-dimensional stroke gesture recognizers between 1991 and 2019. This section systematically describes each identified recognizer by (1) its motivations, (2) its underlying algorithm, (3) its satisfied properties, and (4) the evaluation of its performance. Each algorithm is divided into three parts to facilitate their understanding and their comparison in the next section: (1) the pre-processing applied to all gestures provided as input, (2) the gesture representation used by the algorithm (*e.g.*, a series of points or vectors), (3) and the classification scheme (*i.e.*, how two gestures are compared to each others and their similarity is estimated).

2.1 GRANDMA (Rubine, 1991)

Until 1991, most gesture-based applications were developed with hand-crafted recognition modules or hard-coded toolkits, making these recognizers difficult to create, to modify, and to maintain. GRANDMA [56] significantly departed from this tradition by offering an object-oriented toolkit that extends a direct manipulation UI with gesture-based interaction, thus combining gesturing and direct manipulation. GRANDMA automatically creates a gesture recognizer from template gestures provided by the developer, eliminating the need for hand coding and reducing the effort involved in creating gesture-based UIs. The recognizer, known as Rubine, was later extended with an automatic feature selection [12] to improve recognition rates for a given problem.

2.1.1 Algorithm. GRANDMA automatically produces a 2D uni-stroke gesture recognizers from provided examples for each gesture class. Given G gesture classes specified by example(s) and a candidate gesture c, the objective is to determine the class to which c belongs (*i.e.*, the class whose templates are most like c). For each candidate gesture c, the recognition module runs as follows:

(1) **Preprocessing.** Jiggle is removed from the templates and the candidate gestures by discarding any input point within 3 pixels of the previous point.
(2) **Gesture representation.** A feature vector f is extracted from each gesture with 13 features empirically determined by the author to work well on a number of different gesture sets [56]. These features involve mathematical notions such as the cosine and the sine of angles, the distance between points, the length, speed and duration of the gesture, and the bounding box and its diagonal.
(3) **Classification.** The candidate is classified as one of the possible gesture classes via a linear evaluation of its feature vector over each gesture class' evaluation function. The training problem is to determine the weights associated to each feature in these evaluation functions. For each gesture class, a closed formula computes the weights based on the available templates. Training is performed offline before the classification of any candidate. A linear classifier always returns one gesture class for a candidate. Misclassification could be preferred over rejection in applications with good undo facilities (*i.e.*, if undo is quick and easy, the time spent retrying the gesture will dominate over the effect of a misclassified gesture). The Mahalanobis distance [22] determines the number of deviations a candidate is away from the mean of its corresponding class and the recognition module then rejects gestures appearing as obvious outliers.

2.1.2 Properties. GRANDMA generates a 2D uni-stroke gesture recognizer given templates recorded by a mouse. It recognizes only uni-stroke gestures to avoid the segmentation problem of multi-stroke gesture recognition [60]. The templates should reflect any desired variance in size and/or orientation of the gesture since the generated recognizer is *scale-* and *rotation-sensitive*. The recognizer can be extended with *eager recognition* (*i.e.*, a gesture in progress is recognized as soon as it becomes unambiguous) and *multi-touch recognition* (*i.e.*, by treating a multi-touch input as a multi-path data, the algorithm can be applied to each path individually and the results are combined using a decision tree to classify the multi-touch candidate). These extensions were not evaluated.

2.1.3 Performance. GRANDMA was evaluated on 10 gesture sets whose templates were recorded by the author himself. For GSCORE, an editor for musical scores, fifteen templates per class are adequate for a gesture set holding fifteen classes to achieve a recognition rate of 98%. The training and classification times ranged from 0.2 to 5 ms on a DEC MicroVAX II and the performance improved by a factor of 12 on a DEC PMAX-3100, which means that training and classification are fast enough to be performed in response to user input.

2.2 xstroke (Worth, 2003)

Xstroke [86] introduced pen uni-stroke gesture recognition in two regions: anywhere on the screen, directly on top of the application UI or inside the region specifically tailored for gesture input on a handheld device. A similar recognizer also works well with a trackball [84] used for producing gestures.

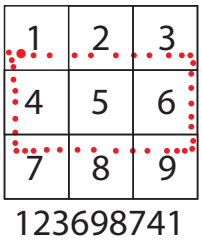


Fig. 2. The sequence of digits generated for a given rectangle. xstroke produces a regular expression describing all recognizable rectangles by combining this sequence with the sequences extracted from all available templates.

2.2.1 Algorithm. Xstroke extends libstroke [1], a stroke translation library that interprets gestures as commands in a program. The recognition algorithm runs as follows:

(1) **Preprocessing.** The template and the candidate gestures' points are linearly interpolated using a Bresenham algorithm to avoid inaccuracies in sampling and speed.
(2) **Gesture representation.** Xstroke implements a highly-featurized representation by constructing a $n \times n$ grid with cells identified. When n=3, the cells are numbered from 1 to 9 centered on the gesture. This grid is scaled independently in the x and y dimensions to fit the bounding box of the gesture. To avoid the deformation of wide or large strokes, the grid is fit to the smallest square containing the gesture whenever the bounding box is more than four times larger in one dimension than the other. The gesture's feature values are determined by traversing the gesture points in time order and recording the number of each grid cell as they pass through it. Fig. 2 illustrates the sequence of digits generated for a rectangle. The recognizer is trained offline by merging the templates' sequences of digits into a regular expression for each class to supposedly cover most users' variations for the corresponding gesture class.
(3) **Classification.** The candidate gesture is mapped to an anchor 3×3 grid numbered from 1 to 9. This second grid is not scaled to fit the bounding box of the gesture but is scaled to cover the entire gesture area, and the center (fifth) cell is enlarged to fill most of the grid area. The anchor grid is used to locate the gesture on the touchscreen to ensure translation-variance (*e.g.*, a gesture performed in the top left corner of the screen will be differentiated from a gesture performed in the bottom right corner). The candidate's sequence of digits is compared to the regular expressions generated during the training session. The gesture class of the regular expression that matches exactly the candidate sequence of digits is returned. Since stroke gestures could vary in orientation depending on their angle on a movable handheld device, rotation-invariance is ensured by applying an orientation correction factor to compensate for any slant. The recognizer adapts to the way the end user records gestures: the correction factor is computed and modified after each successful recognition.

2.2.2 Properties. xstroke is a *2D uni-stroke pen* gesture recognizer satisfying *scaling-* and *rotation-invariances*, with *translation-variance* on-demand due to its anchor grid applied on any candidate gesture at recognition time.

2.2.3 Performance. The algorithm was evaluated on a gesture set containing 82 gesture classes (*e.g.*, all printable ASCII characters such as letters, digits, and punctuation marks, gestures for full keyboard replacement such as backspace and return) input by the author to achieve a 95% recognition rate. No information is reported regarding the testing procedure and the execution time.

2.3 SHARK (Zhai and Kristensson, 2003)

SHARK (SHortand Aided Rapid Keyboarding) [89] is a uni-stroke gesture recognizer for text input: it associates a shorthand symbol for each word according to its movement pattern on an optimized stylus keyboard. The movement pattern is formed by the trajectory from the first to the last letter of the word on the keyboard, thus corresponding to a uni-stroke gesture. Since end users tend to remember patterns more easily and some words are connected in stylus keyboards, tapping on each letter key individually can be replaced by crossing over the letters of a word, thus forming a gesture pattern.

2.3.1 Algorithm. SHARK relies on elastic matching [61], which computes the minimum distance between the candidate points and the templates points by dynamic programming. The template points correspond to the ideal shape of a word built by connecting the centers of the word's key letters on the ATOMIK keyboard. The algorithm works as follows:

(1) **Preprocessing.** The sets of points are filtered and normalized in scale.
(2) **Classification.** The distance between the two sets of points is computed by elastic matching. To resolve any ambiguity between similarly-shaped words (*e.g.*, the words "do" and "no" share identical shapes on the ATOMIK keyboard), the algorithm outputs the N-best list of template words.

2.3.2 Properties. SHARK is a *2D uni-stroke pen* gesture recognizer satisfying *scaling-* and *translation-invariances* on an ATOMIK keyboard.

2.3.3 Performance. A 5-session experiment was carried out on a Wacom ET-0405-U tablet equipped with a pen. Six volunteers were paid to test the usability and the learnability of shorthand gestures as a text entry method based on 100 words list, made up with 50 words selected from the top 100 most common words in the British National Corpus and 50 words selected from the top 100 to the top 300 most common words. The recognition rate and execution time were not reported.

2.4 SHARK^2 (Kristensson and Zhai, 2004)

SHARK [89] faces two serious limitations: end users alternate between two modes of writing in order to learn the shorthand gestures of each word, and its word list is limited in size (*i.e.*, a 100-word lexicon). SHARK^2 [35] eliminates the need to alternate between the two modes of input and extends the lexicon to 10,000 words.

2.4.1 Algorithm. Recognizing 10,000 templates is a challenge that cannot be solved using shape information only. Therefore, SHARK^2 was developed as a multi-channel recognition system such that the collective information from the multiple channels can separate the shorthand gestures sufficiently as follows:

(1) **Preprocessing.** The candidate and the templates are resampled into N equidistant points (*i.e.*, isometricity [67]), then normalized in scale and translation.
(2) **Classification**. SHARK^2 implements a series of consecutive channels whose results are combined to output an N-best list of results:
   (a) An initial component filters out a large number of the shorthand templates based on the start-to-start and end-to-end distances between each template and the candidate. The remaining templates pass through the following channels.
   (b) The *shape* channel introduces "proportional matcher", a linear matching establishing a scaling- and translation-invariant similarity distance between the candidate and each template.
   (c) The *location* channel examines the absolute location of the user's gesture trace on the keyboard by computing the distance between the candidate and the ideal template trace (*i.e.*, the trajectory that connects the centers of the letters that constitute the word).

(d) After shape and location channels are considered, conflicting words may still appear that can be resolved by the language context (*i.e.*, the surrounding words). SHARK^2 uses smoothed bigrams as a language model to solve the issues.
(e) The *integration* channel conciliates the distance scores of the shape and the location channels because these scores are not on a common scale and cannot be directly compared. A confidence score is computed for the word using Bayes rules and based on the confidence scores of both channels. SHARK^2 implements a dynamic weighting scheme between the two channels: if the user's gesture is produced faster than a threshold, then the location channel should yield a poor score since the word was certainly performed on a fast open-loop memory recall, otherwise, the location channel should yield a high score since the word was certainly traced using visual guidance of the virtual keyboard (its location is slightly more meaningful than its shape). The threshold is determined on a per-word basis and depends on the user's completion time, but also on the length and the complexity of the gesture.
(f) Finally, the algorithm outputs an N-best list of results ranked by confidence score.

2.4.2 Properties. SHARK^2 is a *2D uni-stroke pen* gesture recognizer that classifies *partially scaling*- and *translation-invariant* shorthand gestures based on a lexicon. This word list can be extended with user-defined new words.

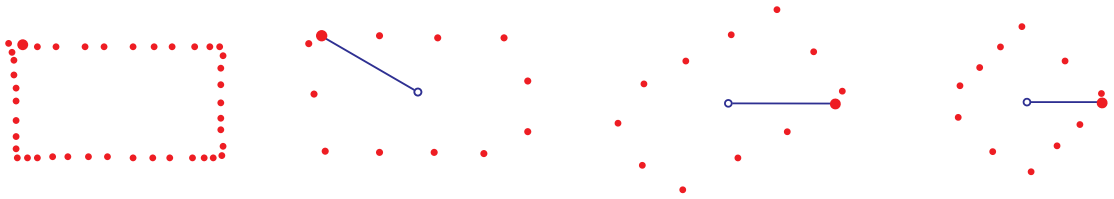
2.4.3 Performance. Although the recognition rates of the algorithm were not reported, the recognizer could search 20,000 shorthand gestures on a 800MHz PIII IBM Thinkpad in 20 to 40ms, which is suitable for interactive shorthand input, thanks to its effective pruning techniques

2.5 $1 (Wobbrock *et al.*, 2007)

$1 [85] is a 2D uni-stroke gesture recognizer supporting the incorporation of gestures into UIs. It can be trained with a variable number of templates, it involves only basic geometry and trigonometry within about 100 lines of code. Many projects and applications used the original algorithm [5, 16, 31, 87, 92]. Quick$ (Quick-Buck) [53] and 1F (One-Feature) [69] are two $1 optimizations that use one or multiple features to determine whether comparing a candidate to a template is necessary, thus reducing the number of comparisons. $3 [33] explore the benefits of the $1 methodology for rapidly classifying 3D gestures.

2.5.1 Algorithm. The recognizer implements a template-matching algorithm to classify a candidate gesture among a set of templates. Each gesture sample results in an ordered set of points. The algorithm determines which set of previously recorded template points most closely matches a set of candidate points, as follows:

(1) **Preprocessing.** Templates and candidate gestures are resampled to N equidistantly spaced points, or isometric points, with some empirical evidence suggesting $32 \leq N \leq 256$. The resampled points are then rotated once around their indicative angle (*i.e.*, the angle formed between the centroid of the gesture and the gesture's first point). After rotation, the gesture is non-uniformly scaled to a reference square. Finally, the gesture is translated so that its centroid is at $(0, 0)$. The four-step preprocessing is illustrated by Fig. 3.
(2) **Classification.** The $1 algorithm computes the Euclidean distance between each pair of points that belongs to the candidate and to any stored template. The sum of these distances describe how similar the gestures are. The classification algorithm outputs an N-best list of templates (*i.e.*, the templates that minimize the sum of the between-point Euclidean distances with the candidate). Despite the initial preprocessing, there is no guarantee that two sets of points will be aligned optimally. The algorithm must, therefore, find the optimal angular alignment between each template and the candidate before computing their between-point Euclidean distance. The $1 recognizer uses the Golden Section Search (GSS) [50]) to find this optimal angular alignment and compares the candidate to any template at each step of the alignment process.



(a) Raw gesture (b) Resampled to N=12 points (c) Rotated (d) Scaled and translated

Fig. 3. Four-step preprocessing applied by the $1 recognizer on a rectangle.

2.5.2 Properties. $1 is a *2D uni-stroke pen* gesture recognizer that is resilient to variations in sampling due to movement speed or sensing, and satisfies *rotation-, scaling-, and translation-invariance*, which is referred to as RST-invariance. Consequently, $1 cannot distinguish gestures whose articulations depend on specific orientations, aspect ratios, or locations. Horizontal and vertical lines are also abused by non-uniform scaling. Fig. 3d highlights these limitations on a rectangle: it is severely deformed by non-uniform scaling.

2.5.3 Performance. The algorithm was tested against GRANDMA's classification module and Tappert's cursive recognizer based on Dynamic Time Warping (DTW) [61] using a dataset collected during a controlled study. This dataset counts 16 uni-stroke gesture classes performed 10 times each at 3 different speeds (*i.e.*, slow to be as accurate as possible, medium to balance speed and accuracy, and fast to draw as quickly as possible) performed by 10 subjects on a Pocket PC using a pen-sized plastic stylus, thus totalling 4,800 templates. Rubine and DTW were extended to perform $1's rotation, scaling, and translation procedures. $1, with a recognition rate of up to 99.02%, and DTW, with 99.15%, were significantly more accurate than GRANDMA, achieving a rate of 92.83% under the same condition. The number of templates and the input speed had a significant effect on recognition rate. Finally, DTW ran noticeably slower than GRANDMA and $1.

2.6 Protractor (Yang Li, 2010)

Template-based recognizers can be time- and memory-consuming as they load all templates, then compare them one by one with any candidate. Protractor [40] is a recognizer suitable for mobile computing with limited processing power and memory as it mitigates both issues. The techniques implemented in Protractor were also used for recognizing 3D gestures [34], and for extending the $-family with the $N-Protractor [4] algorithm, a combination of Protractor, and $N [3] described in subsection 2.7.

2.6.1 Algorithm. Protractor is similar to $1 with differences in terms of rotation, scaling, and dissimilarity score:

(1) **Preprocessing.** Template and candidate gestures are resampled to a fixed number N of equidistantly-spaced points, then translated so that its centroid is at $(0, 0)$. If a template is rotation-invariant, then it is rotated around its centroid by its indicative angle, defined as the direction from the centroid to the first point of the resampled gesture. Otherwise, the indicative orientation of the gesture is aligned with the closest direction of the eight major compass orientations, *i.e.* N, NE, E, SE, S, SW, W, and NW as in [18].
(2) **Gesture representation.** Protractor acquires an equal-length vector for each template and the candidate: $v = [x_1, y_1, x_2, y_2, \ldots, x_N, y_N]$. If N=16, the algorithm outputs a 32-element vector for each gesture.
(3) **Classification.** The matching algorithm then searches for the template that is most similar to the candidate. For each pairwise comparison between a template t and the candidate c, Protractor computes the inverse cosine distance of their vectors, v_t and v_c. The inverse cosine distance describes the dissimilarity between t and c: it finds the angle between two vectors in an N-dimensional space. Since the indicative angle is only an approximate measure of a gesture's orientation, the algorithm employs a closed-form solution to find a

rotation that minimizes the angular distance. This solution replaces the Golden Section Search used in $1 to find the best angular alignment between two gestures. See [40] for the mathematical demonstration.

2.6.2 Properties. Protractor is a *1D* and a *2D uni-stroke pen* and *finger* gesture recognizer as it does not scale gestures and preserves their aspect ratio. The algorithm satisfies *rotation-, scaling-,* and *translation-invariances*, and tolerates *rotation-variance* if necessary. Moreover, Protractor enables developers to specify how sensitive the algorithm should be to orientation, *i.e.*, whether 1 or up to 8 directions may be allowed for a given gesture class.

2.6.3 Performance. Protractor and $1 were evaluated against two datasets on two computing platforms:

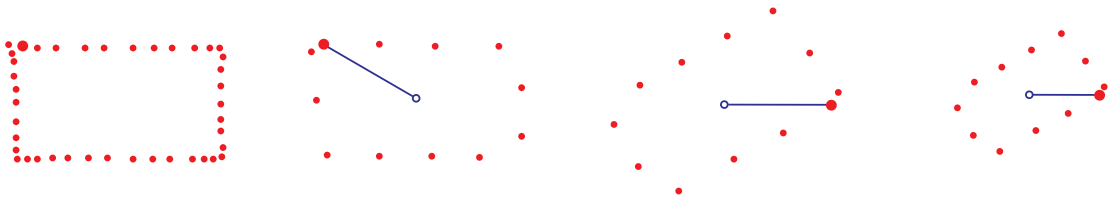
(1) The $1 dataset (4, 800 samples of 16 uni-stroke gesture classes performed 30 times each by 10 participants) performed on a Dell Precision T3400 laptop. Protractor and $1 generated similar error rates in response to different template counts and there was no significant difference between both recognizers in terms of recognition rate. However, Protractor was significantly faster than $1. When 9 templates were used for each of the 16 symbols, $1 took over 3*ms* to recognize a gesture, while Protractor did it in less than 0.5*ms*.
(2) A new dataset (10, 888 samples of 26 uni-stroke Latin alphabet letters) performed by 100 users on their own touch-screen T-Mobile G1 mobile phone. While both recognizers were less accurate on this larger dataset, Protractor performed significantly more accurate and ran significantly faster than $1. When 9 templates were used for each of the 26 gestures, $1 took 1405*ms* and Protractor took only 24*ms*.

2.7 $N (Anthony and Wobbrock, 2010)

Many gestures comprise multiple strokes, which can be issued in multiple ways owing to stroke number, order, and direction. Neither designers nor end users should worry about entering all versions of a multi-stroke gesture, which is a time-consuming and redundant task. The $N algorithm [3] contributes to the incorporation of multi-stroke recognition in UIs. The algorithm is a significant extension to $1 that automatically generates all variations of a multi-stroke. The recognizer has been used in multiple research projects, such as for drawing eyes-free touch gestures on the hand palm [80]. $N-Protractor [4] extended $N with Protractor's closed form solution to find the optimal angular alignment between a candidate and a template. It achieved similar or slightly lower recognition rates than the original $N, but significantly reduced its execution time.

2.7.1 Algorithm. $N is similar to $1, except that all possible permutations of a multi-stroke gesture are generated to handle variations in articulation, and that two optional optimizations fasten the recognition of multi-strokes:

(1) **Preprocessing.** The templates and the candidate gestures are equidistantly resampled, rotated around their indicative angle, scaled uniformly, and translated so that their centroid is at (0, 0). When $1 mistreats 1D gestures because of non-uniform scaling, $N performs a uniform scaling instead. Fig. 4 illustrates how $N pre-processes a rectangle.
(2) **Gesture representation.** A multi-stroke gesture is interpreted as a uni-stroke where parts of the gesture are made away from the sensing surface, *i.e.*, by following the user's hand through the air. To be an effective prototyping tool, $N enables designers to define only one multi-stroke gesture from which all possible uni-stroke permutations and possible combinations of stroke order and direction are generated.
(3) **Classification.** The candidate is compared to all possible permutations of the stored templates and their dissimilarity score is computed as the sum of their between-point Euclidean distances. The score for a template is the best of its uni-stroke permutation' score, and the score for the candidate is the best of the templates' score. During the matching process, $1 operates with full rotation-invariance, meaning that the orientation of a template or a candidate is simply ignored. Similarly to $1, $N relies on the Golden Section Search (GSS) to find the optimal angular alignment between two gestures. It provides the option of bounding rotation-invariance on a per-gesture basis by flagging the rotation-variant templates at design time. These rotation-variant gestures are rotated back to their original angle during the pre-processing step, and the



(a) Raw gesture (b) Resampled to 12 points (c) Rotated (d) Scaled and translated

Fig. 4. Four-step preprocessing applied by the $N recognizer on a rectangle.

GSS technique searches from there instead. Creating all uni-stroke permutations of a multi-stroke, however, results in a combinatorial explosion. To mitigate this issue, two speed optimizations are introduced. First, comparing only uni-stroke permutations whose start directions are "about the same" reduced uni-stroke comparisons by 79.1% and improved accuracy by 1.3% during the evaluation procedure. Second, restricting comparisons of candidates to templates that comprise the same number of strokes, while skipping all others, reduced comparisons by 10.4% and improved accuracy by 1.7%.

2.7.2 Properties. $N is a *1D* and *2D multi-stroke pen* gesture recognizer holding in 240 lines of code. It satisfies *translation-* and *scaling-invariance*, and *on-demand rotation-invariance* to recognize more symbols. The recognizer is also *articulation-invariant*: it is not sensitive to the number of strokes, to their order, and to the direction of these strokes as it generates all possible permutations of a uni-stroke. The algorithm cannot reason about gesture features and collisions are possible if the optional number-of-strokes optimization is used, *e.g.*, the letter "z" and the math symbol "=" could not be distinguished if produced with two strokes.

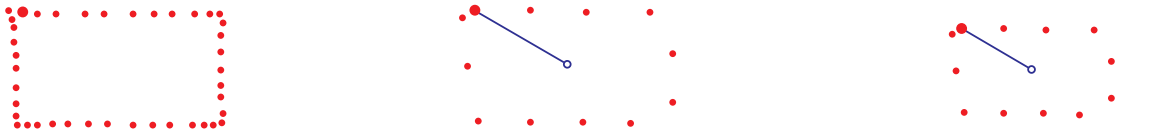
2.7.3 Performance. $N was tested against the $1 dataset and a new *in situ* study in which pen gestures were provided by 40 middle and high school students aged 11-17 using a Tablet PC. Each student copied 45 equations supplying gestures for 20 distinct algebra symbols. The final gesture set comprises 15,309 gestures, with 70% uni-strokes and 30% multi-strokes, of which 13.4% were 1D gestures. The original algorithm reached a recognition rate of 96.6% on the algebra symbols with 15 templates, and a rate of 96.7% on the $1 dataset with 9 templates. $N performed poorer than $1 on its own gesture set. Considering both speed optimizations increased the accuracy and decreased the recognition time of $N.

2.8 $P (Vatavu *et al.*, 2012)

The $N-Protractor, treating a multi-stroke gesture as a uni-stroke obtained by connecting individual strokes performed "through the air", suffers from significant memory and execution costs. As stroke order and direction can differ among users drawing the same gesture, all possible uni-stroke permutations of a multi-stroke need to be generated. Most $N limitations and its extension $N-Protractor come from reasoning about gestures in terms of a chronological order of drawn points, which enforces a predefined order for strokes and points within each stroke. These concerns become irrelevant for $P [73], a multi-stroke gesture recognizer that avoids the storage and computation complexity of $N-Protractor by representing gestures as "clouds of points". Small wearable devices, such as the touch-enabled textile GestureSleeve [57], benefit from such improvements.

2.8.1 Algorithm.

(1) **Preprocessing.** The template and the candidate gestures are equidistantly resampled, scaled uniformly, and translated so that their centroid is at (0, 0). Fig. 5 illustrates $P's three-step preprocessing on a rectangle.



(a) Raw gesture (b) Resampled to 12 points (c) Scaled and translated

Fig. 5. Three-step preprocessing applied by the $P recognizer on a rectangle.

(2) **Gesture representation.** Each gesture is represented as a cloud of points, which is an unordered set of points, invariant to stroke order and direction.
(3) **Classification.** Since gestures are represented as clouds of points, $P does not match points in their sequential order. Instead, for each point in the first cloud, the algorithm finds the closest point from the second cloud that has not been matched yet. Each matching is weighted with a confidence score in [0..1]. The first matching is bound to the higher confidence score and confidence keeps on decreasing for the next matches. As the algorithm progresses, few options remain for the rest of the points from the first cloud when searching their closest pair into the second. The closeness of points is computed by their Euclidean distance and the sum of these weighed between-point Euclidean distances correspond to the dissimilarity score of the entire matching. This procedure is repeated for each template cloud: the candidate is matched to the template, then the template is matched to the candidate. The minimum score of these two matching expresses the dissimilarity score between the candidate and the template. The most similar template (the smallest distance) is considered as the best match for the candidate and is returned by the algorithm.

2.8.2 Properties. $P is a *1D* and *2D uni-stroke* and *multi-stroke pen* gesture recognizer satisfying *scaling-*, *translation-* and *articulation-invariances* (by discarding the gesture timeline, the algorithm becomes insensitive to the number of the strokes, their order, and their direction). $P is, however, *rotation-variant* as the clouds of points are oriented and the GSS technique is no longer used. The recognizer needs less memory than $N and $N-Protractor as it does not compute all permutations of a multi-stroke, thus decreasing memory usage from $O(T \times S! \times 2^S)$ to $O(T)$ with S strokes in a multi-stroke, and T templates. Finally, $P runs in $O(N^{2.5})$ with N resampling points.

2.8.3 Performance. $P was tested on the $N-MMG dataset (3,200 templates of 16 uni-stroke and multi-stroke gesture classes performed by 20 participants). The recognizer was compared against $N and the Hungarian algorithm [46] previously determined as the ideal recognition technique for matching two clouds of points. $P's recognition rate was not significantly different from the rate of the Hungarian algorithm for user-dependent scenario (*i.e.*, when candidates and templates are performed by the same user). However, significant differences were detected between the two recognizers for user-independent scenario (*i.e.*, when candidates and templates are performed by different users), with $P exhibiting higher recognition rates. $P achieved a maximal rate of 99.4% with more than 5 templates per class in the user-dependent scenario and a maximal rate of 99.5% with 10 participants in the user-independent scenario. $P ran 10 times faster than the Hungarian algorithm and 15 times slower than $N for a sampling rate of N=96, which was determined as the optimal rate for $N. However, $P and the Hungarian algorithm delivered better recognition rates than $N when gestures were down sampled to N=32 points, reducing the execution time from 32ms to 5.5ms. Finally, $P was evaluated against $1 with similar recognition rates than its uni-stroke predecessor for both user-dependent (99.3% for $P, and 99.5% for $1) and user-independent scenarios (96.6% for $P and 97.1% for $1).

2.9 SED-NE (Zhang et al., 2013)

The computational cost of Nearest-Neighbor Classifiers (NNC) [22] lies approximately in the product of two factors: the number of templates and the average cost of each comparison. SED-NE (Squared Euclidean Distance Nonlinear Embedding) [91]) speeds up the nearest-neighbor search using the approach proposed in [27, 28]. First, it filters the training set at classification time. Second, it accelerates the distance calculation. The main idea is to embed data vectors into a low-dimensional space in which the recognizer can compute a lower bound distance between a candidate and a template. This lower bound enables the algorithm to reject some templates at classification time, thus reducing the number of comparisons.

2.9.1 Algorithm. SED-NE significantly differs from other nearest-neighbor classifiers:

(1) **Preprocessing.** The templates and the candidate are resampled to N equidistantly-spaced points, then uniformly scaled to fit in a 256×256 square, and finally translated so that their centroid is at $(0, 0)$.
(2) **Gesture representation.** Each gesture can be represented as a $2N$-dimensional vector: $v = [x_1, x_2, \ldots, x_N, y_1, y_2, \ldots, y_N]$. The dissimilarity score of two gestures corresponds to the squared Euclidean distance of their $2N$-dimensional vectors. The algorithm computes the mean μ and the variance σ^2 of these vectors. By applying the Cauchy-Schwarz inequality, the squared Euclidean distance of two gestures is always greater or equal to the squared Euclidean distance in the low-dimensional space based on (μ, σ), and computing distances in this embedded space is more efficient than in the original one. Therefore, this new distance is faster to compute and can be used as a lower bound for the squared Euclidean distance of two gestures. We refer to [27, 28] for the mathematical proofs.
(3) **Classification.** The matching algorithm traverses each template twice. In the first traversal, the lower bound of the distance from each template to the candidate is calculated in the embedded space and stored. The template that minimizes the lower bound is recorded and set as the initial candidate's nearest neighbor. The initial minimum distance corresponds to the squared Euclidean distance between this template and the candidate. In the second traversal, each lower bound is compared with the minimum distance computed so far. If the lower bound is greater than this distance, then the template can be immediately rejected. Otherwise, its squared Euclidean distance with the candidate is computed. If this new distance is smaller than the minimum distance found insofar, the template becomes the new candidate's nearest neighbor and the minimum distance is updated until completion of the second traversal. The template with the smallest squared Euclidean distance in the end is considered as the result of the classification.

2.9.2 Properties. SED-NE is a *1D* and *2D uni-stroke pen* gesture recognizer satisfying *scaling-* and *translation-invariances*, but remaining *rotation-variant.*

2.9.3 Performance. SED-NE was tested against the $1 gesture set, and compared to $1, and $1-SED. The latter used the squared Euclidean distance as a metric instead of the traditional Euclidean distance. The evaluation was performed in a user-independent scenario. The overall recognition rate of the three recognizers was comparable. The squared Euclidean distance performed slightly better than the standard Euclidean distance, but its superiority was negligible. SED-NE expedites the matching process without compromising the recognition rate: the recognizer outperformed significantly the others by filtering approximately 99% of the training templates in the embedded space, resulting in an average speed-up of over 20 times with a resampling rate N=64 and 12 times with N=32.

2.10 Gestimator (Ye and Nurmi, 2015)

Template-based approaches were observed as performing well when the gestures are relatively simple and unambiguous. In more recent applications, the UI gestures become more complex, consist of multiple sub-strokes, and closely resemble to other gestures, thus making them hard to differentiate. Thus, relying only on the gesture

shape should be enriched by sub-stroke identification. Gestimator [88] is a 2D uni-stroke gesture recognizer that segments gestures into sub-strokes, then matches them by comparing the similarity of constituent strokes.

2.10.1 Algorithm. Gestimator implements two techniques to segment gestures into strokes and compare the constituent strokes of a candidate with those of the templates:

(1) **Preprocessing.** First, the algorithm removes jitter by applying a centered moving average filter on the gesture points. Second, the starting and ending points are removed using a density-based clustering algorithm that iteratively merges the first, resp. the last, points in the trajectory until a consistent movement trajectory is identified. All the points in the detected clusters are removed as noise. Performing noise removal avoids inaccuracies in the gesture tracking, and from subtle, unconscious hand movements. After noise removal, the gestures are normalized by unit scaling and centered so that their centroid is at $(0, 0)$.
(2) **Gesture Representation.** An adaptive segmentation identifies peaks in the gesture curvature as a unified measure for detecting changes in speed and direction. Any segmented gesture is divided into sub-strokes. For example, a uni-stroke rectangle is segmented in four strokes, one for each side.
(3) **Classification.** The sub-strokes of a candidate are compared against those of templates. Due to the variations in articulation, two segmentations of the same class can differ from one template to another. To ensure high recognition, an accumulative matching is considered where subsequent strokes are merged if they produce higher similarity than the individual strokes alone. In this way, a rectangle composed of three sub-strokes can still be matched against a rectangle composed of four sub-strokes. The similarity between two gestures is based on Dynamic Time Warping (DTW) [61], but the metric to compute this distance is not mentioned.

2.10.2 Properties. Gestimator is a *2D* and *3D uni-stroke pen* and *hand* gesture recognizer satisfying *scaling-* and *translation-invariances* thanks to its preprocessing steps.

2.10.3 Performance. The recognizer was evaluated against the $1, Protractor, and SHARK^2 recognizers on three datasets: (1) the $1 dataset, (2) the EDS-1 dataset (18 uni-stroke gestures performed 20 times each by 14 participants), and (3) an "authentication" dataset (13 uni-stroke mid-air gestures performed 6 times each by 10 participants using a Microsoft Kinect, these gestures consist of 7 straight line segments and additional swirl strokes that represent a sequence of seven digits on a virtual keyboard). In the user-dependent scenario, all the algorithms achieved a high recognition rate on the simple pen gestures, but their performance decreased on the authentication dataset, which contains complex and ambiguous gesture pairs. Gestimator had the best overall performance, with a recognition rate of 98.15%, and the performance differences with the baseline methods were statistically significant for all datasets. Gestimator was the slowest overall recognizer, reaching an execution time of $82ms$. Protractor was the fastest overall, reaching $28ms$. In the user-independent scenario, the performance of all algorithms suffered. Gestimator performed less accurately than SHARK^2 when the templates from only one or two users were available. However, Gestimator started to outperform its competitors when templates from three or more users were available.

2.11 Penny Pincher (Taranta and LaViola, 2015)

Template-based approaches increase their recognition rate with the number of templates, at the cost of a higher latency: the more templates, the more comparisons are required to output the N-best list of results. Penny Pincher [62] is a 2D uni-stroke recognizer maintaining high accuracy within a time constraint. The algorithm computes a between-point vectors distance instead of a between-point Euclidean distance. The computations are reduced to mere additions and multiplications, avoiding expensive calls to geometric functions such as sine, cosine, and matrix inversion. In [48], Pittman et al. suggest implementing prototype (template) selection instead

of comparing the candidate to all stored templates, to further reduce the recognition time and the error rate of Protractor, $N-Protractor, and Penny Pincher.

2.11.1 Algorithm. In order to evaluate more templates than other recognizers within the same execution time, Penny Pincher avoids calling expensive functions, and rotation, scaling, and translation operations during the preprocessing. The recognizer works as follows:

(1) **Preprocessing.** The candidate and the templates are resampled to N equidistant points: $p_0 \dots p_{N-1}$.
(2) **Gesture representation.** The resampled gestures are converted into a series of between-point vectors of length $N-1$: $v = [p_1 - p_0, p_2 - p_1, \dots, p_{N-1} - p_{N-2}]$.
(3) **Classification.** The dissimilarity score of two gestures is computed as the sum of the angles between their corresponding between-point vectors. Note that a perfect score is $N-1$ because the normalized dot product is 1 for identical vectors. Penny Pincher finds the template that maximizes the dissimilarity score with the candidate and outputs an N-best list of results.

2.11.2 Properties. Penny Pincher is a *2D uni-stroke pen* and *finger* gesture recognizer. Its between-point vector approach is *scaling-* and *translation-invariant*. Like other recognizers, it is not *rotation-invariant* and the templates should consider all relevant variations in angular displacement for a given gesture class.

2.11.3 Performance. Penny Pincher was tested against six datasets: $1-GDS, SIGN, EDS-1, EDS-2, $N-MMG, and UJI (see their comparison in section 3.4). On a MacBook Pro with a 2.4GHz Intel Core i7 processor, 8GB of 1333MHz DDR3 memory, results were reported in a user-independent scenario in comparison to $1, $N, Protractor, $N-Protractor, and 1¢ [25], a handwriting gesture recognizer[1] that shares many similarities with the $-family. Considered too slow for uni-stroke recognition, $P was not included. Penny Pincher reached equivalent or better results than all considered recognizers on the uni-stroke datasets, *i.e.*, $1-GDS, SIGN, EDS-1, and EDS-2. With 10 templates loaded, it achieved a recognition rate of 99.89% on EDS-1, 97.8% on SIGN, and 98.7% on $1-GDS. With 9 templates loaded, it achieved a recognition rate of 99.89% on EDS-2. Penny Pincher did not perform equally as multi-stroke recognizers for multi-stroke datasets (*i.e.*, $N-MMG and UJI), yet it performed reasonably. With 10 templates loaded, it achieved a recognition rate of 92% on MMG. UJI stands out as the most complex dataset as its gestures are very similar and only 2 templates are available for each class. Penny Pincher achieved a recognition rate of 58.47% on this dataset with 1 template loaded per gesture class, which is in line with $1 and Protractor and better than $N and 1¢. In terms of execution speed, 1¢ was the fastest recognizer for comparing 2 gestures in 25*ns*, yet its recognition rate was well below the other recognizers for all 6 datasets. Penny Pincher was also very fast, achieving each comparison in 33*ns*. Given a time constraint to process as many templates as possible, Penny Pincher achieved considerably lower error rate than its predecessors, except for the MMG and the UJI multi-stroke datasets. With this apparatus, Penny Pincher processed approximately 30.3 templates per ms, whereas Protractor –considered as the second most accurate recognizer within the time constraint– was limited to 5.7 templates per ms.

2.12 $P+ (Vatavu, 2017)

Touch-screen devices are less accessible to people with moderate to severe visual impairments, such as with blurred vision, faded colors, and blind spots. People with low-vision face different issues than blind people, primarily as they rely on their visual abilities during their everyday activities. Significant differences in gesture articulation exist between people with low-vision and people without any visual impairment [71]. Since these differences negatively influence the recognition rate of most recognizers (*e.g.*, $1, $N, $P), $P+ [71] is a 2D multi-stroke recognizer that extends $P and remedy these limitations.

[1] According to the exclusion criteria specified in section 3, this algorithm for handwriting recognition was not considered.

2.12.1 Algorithm. $P+ brings two changes on how $P matches the cloud points of the candidate and the template:

(1) **Preprocessing.** Similarly to $P, the templates and the candidate are resampled to N equidistant points, scaled uniformly, and translated so that their centroid is at $(0, 0)$.
(2) **Classification**. $P's matching algorithm is extended so that each point from the first cloud is matched to its closest point in the second cloud, even if that closest point has already been matched before. The weights (confidence score) bound to each match are now superfluous as every point from the first cloud is free to use any point from the second cloud. The standard Euclidean distance, used as a dissimilarity score between two points, now takes into account the difference in turning angles. Bending error measures showed that these angular variations were important enough to discriminate two gestures. The difference in turning angles is computed from three consecutive points to remain independent to stroke direction. The angles are normalized in the interval [0..1].

2.12.2 Properties. $P+ is a *1D* and *2D uni-stroke* and *multi-stroke finger* gesture recognizer preserving $P's properties, *i.e.*, *scaling-, translation-*, and *articulation-invariance*. The $P+'s cloud matching runs faster than its predecessor, *i.e.*, from $O(N^{2.5})$ to $O(N^2)$ with N resampling points, as starting over the matching process with different starting points is no longer needed.

2.12.3 Performance. $P+ was tested with a new dataset acquired with low-vision subjects. This dataset collected 2,445 uni-stroke and multi-stroke samples for 12 gesture classes by 20 participants, among them were 10 with various visual impairments, using a Samsung Galaxy Tab 4. Originally, $P was 12.1% less accurate for gestures produced by low-vision users. $P+ reduced this gap to 1.2%. Indeed, $P+ reached a recognition rate of 94.7% for low-vision users in a user-independent scenario. At its best, $P+ culminated to a recognition rate of 98.2% with templates trained from 8 participants, each providing 8 templates per class. Even for users without visual disturbances, $P+ proved to be more accurate than $P (98.3% vs. 95.6%). In terms of execution time, $P+ was 3 times faster than the original $P in the same scenario.

2.13 Jackknife (Taranta *et al.*, 2017)

Most recognizers analyzed insofar focus solely on one application domain or one input device. General methods that can be adapted to one or many input modalities have received little or no consideration. Moreover, many experiments assume that recognizers are tested on already segmented gestures to avoid issues raised by continuous data streams. Jackknife [64] is a general-purpose gesture recognizer exploiting a collection of recognition techniques to handle various scenarios, including 2D stroke gesture recognition.

2.13.1 Algorithm. Jackknife is a multitool equipped with many functionalities. Dynamic Time Warping (DTW) with the inner product of gesture path direction vectors defines a local cost function. Correction factors are computed that inflate the scores of dissimilar gestures, synthetic gestures are generated for learning a rejection criteria. These techniques are combined together as follows:

(1) **Preprocessing.** Jackknife does not require any preprocessing of candidate and template gestures.
(2) **Gesture representation.** The recognizer represents gestures as a set $\vec{P}$ of unit length direction vectors, referred to as the gesture path direction vectors:

$$\vec{P} = \left\{ \left(\vec{p}_i = \frac{p_{i+1} - p_i}{|p_{i+1} - p_i|} \right) \mid \forall i \in \{1, \ldots, n-1\} \right\} \tag{1}$$

(3) **Classification.** The similarity between a candidate and a template is computed by DTW, which allows local warping of two time-series in order to find their best alignment.
   - The most frequent local cost function associated to DTW remains the squared Euclidean distance over z-score normalized sequences. Inspired by Penny Pincher, Jackknife uses the inner product of the gesture

path direction vectors to measure their similarity. To avoid pathological warping (*i.e.*, when a small part of one sub-sequence is inappropriately mapped to a large portion of another), the classification algorithm constraints the amount of warping allowed by 10% of the time series length.

- To prune templates that will obviously not match a query, Jackknife computes a lower bound score. The lower bound can be used to avoid a full evaluation in two cases: when it is worse than a predetermined rejection threshold or when a closer match has already been found. By accessing both positive and negative templates, the recognizer selects a rejection threshold that prevents and minimizes false-negative and false-positive errors. Jackknife assumes that only a minimum number of positive templates are given, as little as one or two per gesture class. Therefore, the algorithm must generate both positive and negative samples. To create negative samples, positive ones are spliced together to create semi-nonsense, noise-like sequences. To create additional positive samples, the gesture path stochastic resampling technique is used (GPSR) [63]. The rejection threshold is determined after the separate scoring with DTW of the synthetic positive and the synthetic negative samples against their seed samples.
- After the score is computed between the candidate and a template, three correction factors are applied to inflate the score of dissimilar gestures. First, the inverse inner product of normalized feature vectors is used. Intuitively, times series that are similar should score near one so that the DTW score inflation is minimized. Then, the component-wise absolute distance traversed by a gesture is computed. After normalization, this correction factor helps to ensure that the contributions to the gesture path for each component of two different samples are similar in measure. Third, the bounding box extents of the concatenated unnormalized direction vectors is calculated. This bounding box correction factor can be useful in compensating for gestures that are mostly similar, except in span.

2.13.2 Properties. Jackknife is a *modality-independent* gesture recognizer able to classify *2D uni-stroke* gestures as well as *3D full-body*, *3D hand* gestures sampled with a Kinect, a Wii Remote, or a Leap Motion Controller. The recognizer satisfies both *scaling-* and *translation-invariances*, but not *rotation-invariance*.

2.13.3 Performance. Jackknife was tested with various input modalities, such as pen, finger, Wii Remote, and Kinect. In the following, we focus on the evaluation of Jackknife with stroke (pen and finger) gestures. The recognizer was tested in a user-independent scenario on the $1 gesture set. When used as the local cost function, the inner product with gesture path direction vectors yielded significantly higher accuracy than the traditional Euclidean distance. Furthermore, Jackknife significantly outperformed $1 and $P in terms of recognition rate with one or two templates. No comparison is given in terms of execution time.

2.14 G-Gene (Carcangiu and Spano, 2018)

The Engineering Interactive Computing Systems (EICS) community formally models gestures and their sub-parts, paying less attention to the accuracy than the Machine Learning (ML) community, which aims to implement highly accurate and robust to noise, suitable techniques for recognizing complex gestures. This formalization counterpart is suitable for providing guidance to end-users in terms of feedback (*i.e.*, to understand which gesture parts have already been issued) and feedforward (*i.e.*, to discover how to conclude the interaction). G-Gene [14] is a stroke gesture recognizer that generates profiled Hidden Markov Models (HMMs) to establish a consensus between two questions: exhibiting a reasonable recognition rate and modelling gesture sub-parts. DEICTIC [15] and DG3 [19] are follow-up works of G-Gene, which achieve similar accuracy in offline recognition, *i.e.*, while the user is performing the gesture at run-time.

2.14.1 Algorithm. Instead of welcoming raw gesture samples as input for training, G-Gene is trained upon the formal model of each gesture class.

(1) **Preprocessing.** First, a UI designer defines a set of simplified expressions to establish the ideal path of the strokes composing a gesture.
(2) **Gesture representation.** Each gesture is formalized as a set of strings that define its relevant articulations. Three expression literals are used for describing gesture paths:
   - A Point defines the starting position of a stroke through its x and y coordinates, denoted as $P(x, y)$.
   - A Line describes a linear movement from the last stroke position in $\triangle x$, $\triangle y$ direction, denoted as $L(\triangle x, \triangle y)$.
   - An Arc represents a curve movement from the last stroke position in $\triangle x$, $\triangle y$ direction. The movement follows a clockwise or a counter-clockwise rotation, respectively denoted as $A_{\circlearrowright}(\triangle x, \triangle y)$ and $A_{\circlearrowleft}(\triangle x, \triangle y)$.

   G-Gene derives the target profiles of a gesture class from the ideal expressions provided by the designer. A target profile consists of a sequence of characters, each of them modelling a specific gesture sub-part according to its local features. Three character types are taken into account during the derivation process:
   - Each term of type Line is associated to the closest compass direction in the compass (*e.g.*, N=North). Each direction is denoted by its corresponding arrow character (*e.g.*, ↑ for North).
   - Each term of type Arc is associated to a rotation direction, either clockwise or counter-clockwise, respectively denoted by the ↻ and ↺ characters.
   - Each term of type Point is linked to the boundary point between 2 stroke sub-parts, by the · character.

   The derivation process consists of two phases. First, each ground term of the ideal gesture expression is associated to its corresponding character. The result is a single profile string that represents the ideal stroke performance in G-Gene. Second, the process generates a set of all possible mutated profile strings from the ideal profile and the designer selects the changes that preserve the gesture shape. In the end, each gesture expression is associated with a set of profile strings that represent all relevant gesture articulations.
(3) **Classification.** At run-time, G-Gene incrementally builds the representation of the candidate: while the user is issuing a gesture, lines and arcs are distinguished by a colinearity test and a character is generated each time this test changes the result. The recognition problem is now restricted to finding the most similar gesture string among the trained profiles. Profiled HMMs determine the alignment between 2 string representations, finding the most likely correspondence between their characters, and they support identifying which parts of a gesture have been recognized (*i.e.*, feedback) and which parts are the most likely completion of the input (*i.e.*, feedforward).

2.14.2 Properties. G-Gene is a *2D uni-stroke pen* gesture recognizer trained not by templates, but by gesture formal expressions. By exploiting local features, feedback and feedforward are supported, including *eager recognition* as the end user is issuing the gesture. The generated profiled HMMs are *locally scaling-invariant* since their definitions rely on the ideal movement direction and not on its size. They also satisfy *translation-invariance*, but not *rotation-invariance*.

2.14.3 Performance. G-Gene was tested on the $1 dataset. Each gesture class was modeled using profile sequences, apart from the checkmark symbol (✓) which is indistinguishable from the letter "V" because the algorithm is locally scale-invariant. The overall accuracy of G-Gene is 88.2%, which is below the results reported in the literature for the same dataset: $1 (97.1%), $P (96.6%), Protractor (95.5%), and $N (95.2%). Two groups of gestures are separated depending on their recognition rate: gestures benefiting from a very high recognition rate (*e.g.*, letter "X", rectangle, delete) and gestures varying between 70% and 85% (*e.g.*, left square bracket, pigtail, circle).

2.15 $Q (Vatavu *et al.*, 2018)

In recent years, small electronic devices such as smart rings, watches, phones, and other embedded or wearable systems are proliferating, yet being considered as "low-resource devices", meaning they have lower processing power and smaller memory than laptops and desktops. Therefore, $P becomes very expensive for these devices

because of its quadratic order complexity. After looking at several methods to reduce $P's execution time, code optimization techniques accessible to any programmer are not sufficient for these devices [74]:

(1) During the matching of a candidate and a template, the algorithm does not loop anymore through all the points of the second cloud of points, but enumerates only the points that have not been matched yet.
(2) During the same matching process, the square root of the Euclidean distance represents a computational overhead, yet only the relative order of distances is needed and not their absolute magnitudes. Therefore, the square Euclidean distance is now used to assess the dissimilarity between two points.
(3) Dividing all the weights by N, the number of sampling points, also represents unnecessary computations that are now removed.

Although speed-ups implied by these code optimizations are observable, this optimized-$P is still far from offering a real-time response. $Q [74] incorporates new algorithmic techniques to further reduce the $P computations, thus enabling real-time stroke gesture recognition for low-resource devices.

2.15.1 Algorithm. $Q reduces the candidate's classification time of the optimized-$P with new techniques:

(1) **Preprocessing.** Like for $P, the templates and the candidate are resampled to N equidistantly-spaced points, scaled uniformly, and translated so that their centroid is at $(0, 0)$. Then, the gesture points are converted into a "look-up table" (LUT), a 2D grid of equally-spaced points computed once for each template. The LUT implements a look-up point-matching technique that searches for the closest point in $O(1)$ time and a lower bound in $O(N)$ at classification time. Empirical evidence showed that a grid of 64×64 point delivered a good compromise between extra memory demands and accuracy.
(2) **Classification.** $Q builds upon the optimized-$P's cloud matching procedure. As soon as the minimum score cannot be improved by the two clouds, $Q implements early abandoning and avoids unnecessary comparisons during the cloud-matching process. Valuable execution time can be saved by calling the cloud distance function only on-demand. For this purpose, $Q implements lower bounding: given two clouds of points, if their lower bound exceeds the minimum score computed so far, then their dissimilarity should no longer be computed since its value cannot be inferior to this lower bound. For each pair of clouds, $Q computes multiple lower bounds, one for each starting points. The look-up point-matching technique is finally operated: given a template's point, the candidate's look-up table returns the closest candidate's point in $O(1)$, thus benefiting from a lower bound in $O(N)$ for each template cloud and the candidate.

2.15.2 Properties. $Q is a *1D and 2D uni-stroke* and *multi-stroke pen* gesture recognizer specifically tailored to low-resource devices, reducing by >97% the computations needed by $P during the cloud-matching process. The algorithm still preserves *scaling-, translation-* and *articulation-invariances*, and remains *rotation-variant*.

2.15.3 Performance. $Q was tested on the $N-MMG dataset acquired for $N (9, 596 samples of 16 uni-stroke and multi-stroke gesture classes, acquired by 20 participants using a stylus or a finger on a tablet PC device). To assess its performance on low-resource devices, $Q was tested on 6 distinct CPU architectures ranging from high-end to low-power, low-area profiles. To reach a recognition rate of > 99% in a user-independent scenario, $P needed about 6s on the fastest CPU (Cortex-A53), more than 8s on the Cortex-A9 and the Cortex-A7, and about 12s on the low-end CPU (Cortex-A5). The optimized-$P considerably saved time: 45% on average and 54.8% for the fastest CPU (Cortex-A53). This optimized version ran the most demanding scenario in 2, 713ms instead of the 5, 838ms required by the original $P. Yet, $Q completed in 82ms only, achieving massive time savings. The recognition rate increased from 98% to 98.7% on average for the user-dependent training, and from 95.7% to 96.7% for the user-independent training. On a very low-end CPU, like the ARM-1136, $P completed a single classification in 115.3ms with 64 templates loaded. However, $Q took less than one second (813ms) to produce the same classification. On the most powerful CPU, *i.e.*, Cortex-A53, $Q showed a 49× speed-up. On the less powerful one, *i.e.*, ARM-1136, it showed a 142× speed-up.

2.16 !FTL (Vanderdonckt *et al.*, 2018)

Most Nearest-Neighbor Classifiers (NCC) [22] require normalizing gestures before classifying any candidate. Normalization can imply resampling, scaling, translation, and rotation. To avoid this preprocessing, !FTL [67] introduces a dissimilarity metric called the "Local Shape Distance" (LSD) computed between two shapes (or bivectors), a shape [43] being referred to as a pair of vectors (a bivector) connecting three consecutive points of a stroke to form an oriented triangle. The LSD mathematically preserves rotation-, scaling-, and translation-invariances.

2.16.1 Algorithm. The !FTL recognizer works as follow:

(1) **Preprocessing.** If necessary, the raw gesture points are linearly interpolated such that the algorithm keeps N interpolated points per gesture sample.
(2) **Classification.** The candidate's interpolated points are compared to all templates' interpolated points via a sequential matching. The dissimilarity score between two gestures is given by the sum of their between-bivector LSDs. Given a candidate's interpolated points $c_0, \ldots, c_N$ and a template's interpolated points $t_0, \ldots, t_N$, the LSD between a candidate's bivector $(\overrightarrow{a}, \overrightarrow{b}) = (\overrightarrow{c_{i-1}c_i}, \overrightarrow{c_ic_{i+1}})$ and a template's bivector $(\overrightarrow{u}, \overrightarrow{v}) = (\overrightarrow{t_{i-1}t_i}, \overrightarrow{t_it_{i+1}})$ is given by the formula:

$$LSD(\vec{a}, \vec{b}, \vec{u}, \vec{v}) = \sqrt{\frac{|\vec{a}|^2|\vec{v}|^2 + |\vec{b}|^2|\vec{u}|^2 - 2\left[(\vec{a}\cdot\vec{b})(\vec{u}\cdot\vec{v}) - (\vec{a}\cdot\vec{v})(\vec{b}\cdot\vec{u}) + (\vec{a}\cdot\vec{u})(\vec{b}\cdot\vec{v})\right]}{|\vec{b}|^2|\vec{v}|^2}} \tag{2}$$

The template closest to the candidate in terms of the sum of their between-bivector LSDs is returned.

2.16.2 Properties. !FTL is a *2D uni-stroke* gesture recognizer satisfying *rotation-, scale-, translation-invariances* after the interpolation of raw gesture points. !FTL is intrinsically not multi-stroke as articulation-invariance is not fulfilled: points are matched in their order of appearance. !NFTL executes two times the !FTL recognizer, one in each direction, thus doubling the execution time while slightly improving the recognition rate.

2.16.3 Performance. !FTL was tested against $1 and $P on the NicIcon dataset [81], which consists of 13 uni- and multi-stroke gesture classes performed 30 times each on average by 35 participants, thus totalling 14,005 templates acquired with a pen. The algorithm was evaluated with two metrics: the local shape distance (LSD) and the normalized local shape distance (NLSD), a version independent on the lengths of the vectors representing gestures. An 13-inch Apple MacBook running a Intel Core i5 2.9GHz processor was used to perform the evaluation in a user-dependent scenario. The algorithms were trained with 29 templates per class and tested with 1 candidate for each gesture class. No significant difference was found for recognition rate between !FTL-LSD, !FTL-NLSD, and $P, and a small significant difference with $1. There was a significant difference though in execution times between all recognizers: !FTL-NLSD was faster than !FTL-LSD, which was in turn faster than $1, itself faster than $P. Indeed, the algorithm runs in $O(N)$, comparing all N points in order of their appearance.

3 COMPARATIVE ANALYSIS

The analyzed recognizers fall into two categories when classifying a candidate: they either adopt a feature-based approach or a template-based approach. Both techniques use multiple metrics to determine the similarity between a candidate and the templates [73]. The accuracy of each recognizer was evaluated by the mean of the recognition rate in different conditions, with specific or more general gesture sets, along with different modalities of input, or input devices, and various users. Some algorithms appear sensitive or not to one or multiple invariance properties. In this section, we conduct a comparative analysis of the differences and similarities of the described recognizers.

Table 1. Comparison according to the classification models, matching algorithms, and dissimilarity metrics.

Recognizer	Model	Matching algorithm	Dissimilarity metric
GRANDMA	Parametric	Sequential	Feature-based linear evaluation
xstroke	Parametric	Sequential	String-based comparison
SHARK	NNC	Elastic	Sum of the between-point Euclidean distances
SHARK^2	NNC	Elastic	Sum of the between-point Euclidean distances
$1	NNC	Sequential	Sum of the between-point Euclidean distances
Protractor	NNC	Sequential	Between-vector inner cosine distance
$N	NNC	Sequential	Sum of the between-point Euclidean distances
$P	NNC	Cloud	Sum of the between-point Euclidean distances
Gestimator	NNC	DTW	– (Not reported)
SED-NE	NNC	Sequential	Between-vector squared Euclidean distance
Penny Pincher	NNC	Sequential	Sum of the between-vector angles
$P+	NNC	Cloud	Sum of the between-point extended Euclidean distances
Jackknife	NNC	DTW	Between-vector inner product
G-Gene	Parametric	HMM	Feature-based evaluation
$Q	NNC	Cloud	Sum of the between-point Euclidean distances
!FTL	NNC	Sequential	Sum of the between-bivector local shape distances

3.1 Classification Techniques

Researchers have used multiple template-matching approaches to recognize 2D stroke gestures. Many recognizers started classifying gestures with the help of heavy machine learning techniques [17], such as Support Vector Machines (SVMs), Hidden Markov Models (HMMs), Conditional Random Fields (CRFs), decision trees, and Random Forests (RFs), because they are ubiquitous for handwriting and sketch recognition. These techniques are highly accurate, robust to noise, therefore suitable for recognizing complex gestures. Nevertheless, they are inappropriate for gesture customization and rapid prototyping of gesture-based UIs. They require a significant number of templates per class to be efficiently trained. Since end users are unwilling to provide more than a few templates per gesture, this procedure becomes tedious and could be replaced by synthetically-produced gestures [39]. They require calibration, training, and fine-tuning, an advanced knowledge in machine learning rarely possessed by practitioners. Table 1 classifies approaches that have emerged for the aforementioned 2D stroke gesture recognizers.

3.1.1 Parametric approach (feature-based). Classifying stroke gestures is intrinsically less complex than recognizing free handwriting. Rubine [56] paved the way for simple gesture recognition by extracting a vector of 13 features for each candidate and template. An inductive learning method then derives a single model for each gesture class in the training set. Any candidate must therefore fit the parametric model of a gesture class to be classified in this class. For these reasons, the Rubine recognizer has been extensively used in many projects and applications, is referenced in dozens of patents, has been extended with more features to improve its recognition rate [12], and generalized to 3D stroke gestures [59]. xstroke [86] is another parametric recognizer: a sequence of features (digits) for each gesture is generated on a first grid; and the location of a candidate on the touch-sensitive screen is differentiated on a second grid. G-Gene [14] is a third parametric recognizer which combines the precision of machine learning and the simplicity of gesture formalization: Hidden Markov Models (HMMs) map a sequence of observations into a corresponding sequence of features (labels). This corresponding sequence, or *profile string*, is extracted from the template for training. An outstanding advantage of the parametric approach is that the size of the training set increases the training time required to extract the models, and hopefully its recognition rate, but keep the recognition time constant: a candidate either fits the model or not.

3.1.2 Nearest-Neighbour Classification (template-based). The members of the $-family, *i.e.*, $1, $N, $P, $P+, and $Q, employ another classification method. A candidate gesture is first preprocessed to avoid ambiguities due to sampling, scaling, position, and rotation. Then, the candidate is compared to all templates to retain the best matching template. This template-based approach does not require any inductive learning, but uses a Nearest-Neighbor Classification (NNC) [22] to measure the (dis)similarity between two gestures. The gesture class of the closest template, *i.e.*, the one that minimizes or maximizes the distance score, is the best match for the candidate and is output by the classifier. The score, usually expressed as a normalized value or a percentage, expresses how confident the classification is. In contrast with parametric recognizers, both the recognition rate and the execution time of a NNC recognizer are expected to increase as the training set grows since the candidate must be compared to more templates.

The $-family and !(N)FTL [67] calculate the distance between two gestures by a sequential or a cloud matching. Sequential matching consists of matching the points in order of their appearance, while the cloud matching ignores this order and considers that a point from the first cloud can be matched to any point from the second cloud. SHARK [89] and SHARK^2[35] use an optionally nonlinear matching called elastic matching, an optimization problem with respect to a linear or nonlinear point-to-point mapping, called 2D warping [65]. Jackknife [64] and Gestimator [88] also adhere to the template-based approach by matching two templates via Dynamic Time Warping (DTW), a generic technique for matching time series data by computing the optimum chronological alignment between two series using dynamic programming and local warping of the time series [20, 64].

3.1.3 Metrics. Almost half the recognizers (6/16) compute the sum of the between-point Euclidean distances to measure the dissimilarity of two gestures. Three algorithms convert gestures into vectors and compute a between-vector score instead: Protractor calculates a between-vector inverse cosine distance, whereas Jackknife prefers a between-vector inner product, and SED-NE the between-vector squared Euclidean distance (*i.e.*, the square root of the Euclidean distance is not computed). Penny Pincher represents each gesture as an ordered set of vectors, then computes the sum of the between-vector angles of two gestures. Finally, !FTL converts gestures into a series of bivectors and measure the dissimilarity between two series of bivectors by calculating the sum of their between-bivector Local Shape Distances. The similarity metric used by Gestimator was not mentioned by its authors. Some studies [66, 68] compared metrics to identify their impact on recognition rate and time.

3.2 Modalities of input

Gestures largely vary in the ways they are performed by end-users and sampled by sensors or input devices. A modality of input defines a language that can be used to interact with a computer system through one of its sensor, *e.g.*, touch, mid-air, and whole-body gestures are three possible modalities of input. This language can be influenced by multiple factors, such as the channel used for sampling gestures, a touch-sensitive surface in our case, or the intermediary used for touching the surface, most frequently a pen or a finger as shown by Table 2. This survey focuses on 2D stroke gesture recognizers, which are primarily aimed at recognizing touch-based gestures, not touch-less gestures. Beyond their ability to recognize 2D gestures, some recognizers also cover 1D and/or 3D gestures, such as Jackknife and Gestimator.

The algorithms were mainly developed for touch gestures as 15 out of 16 algorithms were evaluated with pen and/or finger gestures performed on a touch-sensitive screen. All recognizers are limited to uni-touch gestures, but some have been extended to multi-touch recognition. For example, $P was also tested on a multi-touch dataset [54], thus leading to several key improvements to produce an accurate multi-touch recognizer. GRANDMA has taken into account mouse gestures, while Gestimator and Jackknife expanded their evaluation to hand and full-body gestures performed in front of a Leap Motion Controller and a Microsoft Kinect. $1, as well as SED-NE and Gestimator, cannot classify 1D gestures such as horizontal and vertical lines (which are useful to determine flicks, swipes, and straight lines), as they share $1's preprocessing, which abuses 1D gestures by non-uniform

Table 2. Comparison according to the dimensionality, stroke type, and modalities of input.

Recognizer	1D	2D	3D	Multi-stroke	Tested with ... gestures
GRANDMA	●	●	○	○	Mouse
xstroke	●	●	○	○	Pen
SHARK	●	●	○	○	Pen
SHARK^2	●	●	○	○	Pen
$1	○	●	○	○	Pen
Protractor	●	●	○	○	Pen; Finger
$N	●	●	○	●	Pen
$P	●	●	○	●	Pen
SED-NE	○	●	○	○	Pen
Gestimator	○	●	●	○	Pen; Hand
Penny Pincher	●	●	○	○	Pen; Finger
$P+	●	●	○	●	Finger
Jackknife	●	●	●	○	Pen; Hand; Full-Body
G-Gene	●	●	○	○	Pen
$Q	●	●	○	●	Pen
!FTL	●	●	○	○	Pen

scaling. $N, $P, $P+, and $Q remedy this limitation by implementing a uniform scaling. Jackknife and Gestimator were tested with 2D *and* 3D stroke gesture recognizers.

3.3 Evaluation

Two scenarios, denoted as "user-dependent" and "user-independent", are promoted for evaluating gesture recognizers. The former consists of training and testing the recognizers with gestures performed by the same user. The latter consists of training the recognizers with gestures performed by one or multiple users, and testing with gestures performed by an independent user. The evaluation protocol is consistently conducted:

(1) A set of T templates are used for training the recognizer.
(2) One (independent) gesture sample per gesture class is picked for testing the algorithm. All samples are chosen among those that have not been used before as templates for training. In a user-dependent scenario, the candidates are chosen among the gestures performed by the same participant for training the algorithm. In the user-independent scenario, they are chosen among P independent participants.
(3) This two-step process is repeated R times for each level of P and T. The results are averaged to measure the recognition rate expressed in percentage [%] and the recognition time expressed in milliseconds [ms].

Table 3 depicts the scenarios in which the aforementioned recognizers were evaluated, as well as the number of templates (T) and participants (P) used to train these recognizers. SHARK and SHARK^2 were evaluated only for their abilities to discover and learn easily shorthand gestures when writing with a virtual keyboard, and to write as many words per minute as possible with shorthand gestures. Even if this evaluation scheme is considered as user-independent (*i.e.*, the candidates are tested against ideal templates generated by an algorithm for each word on the virtual keyboard), their recognition rates and execution times stand unreported. GRANDMA and xstroke were exclusively tested in a user-dependent scenario because their datasets were limited to gesture samples performed by their respective author. $P, $P+, and $Q were evaluated in both the user-dependent and the user-independent scenarios with $T = [1, 2, 4, 8]$ templates in the user-dependent scenario, and $T = [1, 2, 4, 8]$ templates per participant from $P = [1, 2, 4, 8]$ participants in the user-independent scenario. SED-NE and Jackknife

Table 3. Comparison according to the evaluation scenarios, and their template and participant counts.

Recognizer	User-dependent	User-independent	Templates (T)	Participants (P)
GRANDMA	●	○	1..99	1
xstroke	●	○	?	1
SHARK	○	○	0	1
SHARK^2	○	○	0	1
$1	●	○	1..9	1
Protractor	●	○	1..9	1
$N	●	○	1..15	1
$P	●	●	1..9	1..19
SED-NE	○	●	1..9	9
Gestimator	●	●	1..5	1..5
Penny Pincher	●	●	1..10	1..60
$P+	●	●	1, 2, 4, 8	1, 2, 4, 8
Jackknife	○	●	1, 2	9
G-Gene	○	●	1	1
$Q	●	●	1, 2, 4, 8	1, 2, 4, 8
!FTL	●	○	29	1

were trained with templates from all participants, except one (independent) participant used for testing. Penny Pincher was simultaneously evaluated in the user-dependent and user-independent scenarios: the recognizer was trained with $T=[1..10]$ templates per class performed by all participants of the same dataset, then one sample per class was chosen to be the candidate among the samples that had not been chosen for training. Moreover, Penny Pincher was evaluated in a cost-effective condition: given a time constraint (from 0μs to 100μs), the recognizer was allowed to process as many templates as possible within the given boundary before testing a candidate sample from an independent participant. Gestimator was evaluated in the user-independent scenario with $T=[1..5]$ templates from $P=[1..5]$ participants. !FTL was evaluated in a user-dependent scenario with 29 training samples per gesture class, a relatively large number of templates compared to its predecessors. In this condition, the algorithm proved highly accurate for multi-stroke gestures, although articulation-invariance is not satisfied, since almost all possible articulations for a multi-stroke were covered by the 29 templates. The evaluation procedure applied to the G-Gene algorithm is particular since it cannot be trained by gesture samples performed by end-users. Instead, G-Gene used the profile sequence entered by the designer for each gesture class (one template, one participant) for training its profile HMMs.

3.4 Datasets

Evaluating a recognizer requires the acquisition of a dataset, from which the evaluator can pick a training and a testing set. A set suited for evaluation should contain at least two templates for each class, one for training and one for testing. Table 4 illustrates the different datasets acquired and used for evaluating the analyzed recognizers. The 3D full-body and hand gesture sets acquired to evaluate Jackknife and Gestimator are considered out of scope. The $N-MMG gesture set was defined in $N [3], but was acquired later for evaluating $N-Protractor [4]. More than half the datasets ($\frac{11}{17}$=65%) were acquired with a digital pen. Others include the participant's finger, usually the index, or a computer mouse. $N-MMG consists of both pen and finger gestures as half the participants performed pen gestures and the other half performed finger gestures. The $1 and $N-MMG's participants performed 10 templates per class at three speeds (*i.e.*, slow, medium, and fast), allowing to measure the impact of speed on recognition accuracy.

Table 4. Datasets.

Dataset	Multi-stroke	Modality	Classes	Templates/class	Participants	Samples	Used
GScore	○	Mouse	30	100	1	1,000	1
Coleman	○	Mouse	11	100	1	1,100	1
Digits	○	Mouse	10	100	1	1,000	1
Letters	○	Mouse	26	100	1	2,600	1
xstroke	○	Pen	82	?	1	?	1
SHARK	○	Pen	100	1	1	100	1
SHARK^2	○	Pen	7,777	1	1	7,777	1
$1	○	Pen	16	30	10	4,800	9
Protractor	○	Finger	26	4-5	100	10,888	1
$N-Algebra	●	Pen	20	≥9	40	15,309	1
$N-MMG	●	Pen; Finger	16	10	20	3,200	4
SIGN	○	Pen	17	25	20	8,500	1
EDS-1	○	Pen	18	20	14	5,040	1
EDS-2	○	Pen	11	20	11	4,400	1
UJI	●	Pen	98	2	60	11,640	1
$P+	●	Finger	12	10	20	2,400	1
NicIcon	●	Pen	14	≥30	33	14,005	1

The datasets also feature a wide range of gesture classes, number of templates per gesture class, and number of participants. If GScore, Coleman, Digits, Letters, xstroke include samples from only one participant, their respective author, others gather samples from multiple subjects and support user-independent testing. SHARK and *SHARK*^{2} also support user-independent testing since the training example for each gesture class is a synthetic ideal template on the virtual keyboard. Finally, some datasets have been used much more frequently: $1 stands as the reference set for evaluating uni-stroke recognition and $N-MMG, for evaluating multi-stroke recognition. Penny Pincher is the recognizer that has been tested with the widest range of gesture sets, four, among which three are uni-stroke and one is multi-stroke.

3.5 Gesture Sets

A gesture set defines all possible gesture classes contained in a dataset. Researchers have used multiple gesture classes for evaluating their algorithms, which we can split into seven categories:

(1) Latin alphabet letters (*i.e.*, a-z, A-Z) and punctuation marks (*e.g.*, exclamation and interrogation points);
(2) Digits (*i.e.*, 0-9);
(3) Geometric shapes (*e.g.*, squares, rectangles, triangles, stars);
(4) Symbols (*e.g.*, mathematical operators, highly-symbolic and/or iconic paths);
(5) Patterns (*e.g.*, undefined paths);
(6) Horizontal, vertical and curved lines;
(7) 1D points (*e.g.*, simple tap, double tap).

Table 5 enumerates and categorizes the gesture classes for the datasets listed in table 4. Their gesture sets mainly consist of letters, punctuation marks, geometric shapes, and symbols. SHARK and SHARK^2 consider the ideal template of 100 words, resp., 7,777, on a virtual keyboard, which we consider as patterns since they merely join the letters of each word on the keyboard. The GScore and the xstroke sets are the only ones considering 1D points, yet no set covers the seven categories together. The xstroke set is unique in that multiple variations coexist for a same class. For example, the "a" letter can be articulated as a cursive letter and as a ∧ symbol. We did not

Table 5. Gesture sets.

Dataset	Gesture classes	Letters	Digits	Shapes	Symbols	Patterns	Lines	Points
GScore	30	9	–	2	2	10	4	1
Coleman	11	–	–	–	2	3	6	–
Digits	10	–	10	–	–	–	–	–
Letters	26	26	–	–	–	–	–	–
xstroke	82	46	9	–	11	10	5	2
SHARK	100	–	–	–	–	100	–	–
SHARK^2	7,777	–	–	–	–	7,777	–	–
$1	16	3	–	4	8	–	1	–
Protractor	26	26	–	–	–	–	–	–
$N–Algebra	20	5	10	–	5	–	–	–
$N–MMG	16	8	–	2	4	–	2	–
SIGN	17	2	–	2	6	–	7	–
EDS–1	18	3	3	3	9	–	–	–
EDS–2	20	–	–	–	–	–	–	–
UJI	97	88	10	–	–	–	–	–
$P+	12	5	–	3	4	–	–	–
NicIcon	14	–	–	–	14	–	–	–

Table 6. Comparison according to the properties of invariance.

Recognizer	Sampling	Scaling	Translation	Rotation	Articulation	On-the-fly	Guidance
GRANDMA	●	○	●	○	○	●	○
xstroke	●	●	on-demand	○	○	○	○
SHARK	●	●	●	○	○	○	○
SHARK^2	●	●	partially	○	○	○	○
$1	●	●	●	●	○	○	○
Protractor	●	●	●	on-demand	○	○	○
$N	●	●	●	on-demand	●	○	○
$P	●	●	●	○	●	○	○
SED-NE	●	●	●	○	○	○	○
Gestimator	●	●	●	○	○	○	○
Penny Pincher	●	●	●	○	○	○	○
$P+	●	●	●	○	●	○	○
Jackknife	●	●	●	○	○	○	○
G-Gene	●	locally	●	○	○	●	●
$Q	●	●	●	○	●	○	○
!FTL	●	●	●	●	○	○	○

consider the variations in our count, but only the gesture classes (*e.g.*, the letter "a" counts for one gesture despite that xstroke considers two possible distinct paths). Finally, the NicIcon set includes only simplified highly-iconic symbols used in emergency systems (*e.g.*, fire, gas, flood).

3.6 Properties

End users produce gestures in lots of different ways: each gesture can be sampled, rotated, scaled, or located differently. To mitigate these effects, some recognizers are invariant to Articulation, Rotation, Scaling, and Translation (ARST-invariant). Table 6 shows the properties satisfied by the recognizers. Obviously, all 2D stroke gesture recognizers are *sampling-invariant*, thus tolerating the variations induced by the hardware (*e.g.*, some pointers face physical limitations) and the software (*e.g.*, applications unequally share computer resources depending on the operating system). In practice, most recognizers start preprocessing templates and candidates with an equidistant resampling (isometricity) or a linear interpolation to avoid the variations due to the sampling device. Sampling by establishing a same time interval between points (isochronic resampling [67]) was not tested.

An *articulation-invariant* recognizer classifies a gesture independently from its number of strokes, their drawn order, and their direction. All uni-stroke recognizers are articulation-variant as they do not support the simultaneous classification of multiple strokes. Multi-stroke recognizers from the $-family are the only articulation-invariant classifiers: $P, $P+, and $Q implement cloud matching and $N generates all uni-stroke permutations of a multi-stroke to satisfy this property. !FTL is sensitive to order and direction and !NFTL is direction-invariant. Although Gestimator breaks down the gesture articulation into sub-strokes, its sequential matching using DTW does not tolerate variations in order and direction of the strokes.

Except GRANDMA, the recognizers are also *scaling-invariant*: they classify gestures independently of their scale. Scaling-invariance should not be confused with size-invariance. In the former case, two rectangles are considered similar if they respect the same ratio in width and in length. In the latter case, two rectangles are considered dissimilar if they differ in width and/or in length. Notice that G-Gene ensures local scale-invariance (*i.e.*, at the stroke level) because it uses local features to represent each stroke.

Translation-invariance, sometimes also referred to as position-invariance, means that a gesture's position on any interactive surface does not influence its recognition or that translation does not influence the process. SHARK, SHARK^2, and xstroke are translation-variant classifiers: the position of a shorthand gesture on top of a virtual keyboard is a fundamental concept for SHARK, and xstroke considers that translation-variant gestures enable using the same gesture for multiple tasks, depending of their position on the surface.

Gestures can be rotated significantly from one user to another depending on the context of use: our writing styles naturally invite us to draw gestures with different slants, we use mobile and handheld devices in different orientations due to the physical arrangement of our human shoulders and arms, or our individual preferences, and some surfaces, like circular tabletops, should support any angle. Therefore, strokes differ in orientation and uncorrected slant can lead to misrecognition. Few recognizers are *rotation-invariant*, most of them augmenting their execution time to find the optimal angular alignment between two gestures. However, !FTL satisfies rotation-, scaling-, and translation-invariance because it uses the Local Shape Distance (LSD), a metric that describes the dissimilarity between two bivectors, remaining insensitive to any change of these properties, without any preprocessing.

4 DISCUSSION

Based on the similarities and differences between recognizers highlighted in the previous section, we provide a decision table for UI prototypers, designers, and researchers who wish to pick the right recognizer for their gesture-based UIs, then we discuss and summarize the domain's next points of interest: how stroke recognition could be extended to new horizons and how to improve the evaluation of actual and future recognizers.

4.1 Decision Table for UI Prototypers, Designers, and Researchers

In section 3, we highlighted differences between recognizers in terms of stroke counts (*i.e.*, uni- and multi-strokes), properties (*i.e.*, sampling-, articulation-, rotation-, translation-, and scaling-invariance), modalities (*i.e.* mouse,

Table 7. Decision table for UI prototypers, designers, and researchers. The recognizers are highlighted in red when they require training with user-dependent samples, in green when they proved accurate when trained with user-independent samples, and in blue when the designer must input the formal definition of all gesture classes instead of providing samples.

Strokes	Invariant to...	Modalities	Recognizers
Uni-	Sampling, Translation	Mouse	**GRANDMA**
	Sampling, Scaling, Translation	Pen	**xstroke**,**SHARK**$^{(2)}$, **SED-NE, Gestimator, Jackknife**
		Pen, Finger	**Penny Pincher**
	Sampling, Scaling, Translation, Rotation	Pen	**$1, Protractor, !FTL**
Multi-	Sampling, Scaling, Translation, Articulation	Pen	**$P+**, **G-Gene**
		Pen, Finger	**$P, $Q**
	Sampling, Scaling, Translation, Rotation, Articulation	Pen	**$N**

pen, and finger), and evaluation (*i.e.*, user-dependent *vs.* user-independent testing). Table 7 suggests a decision table for UI prototypers, designers, and researchers for choosing the right recognizer according to these four criteria. Based on one's gesture set, the table consists of answering the following questions:

- Does the gesture set contain uni-strokes or multi-strokes ?
- What properties of invariance are required for differentiating all gesture classes ?
- Are the end-users supposed to interact with a mouse, a pen, and/or a finger ?
- Do I have available subjects to acquire training samples ? If not, favour user-dependent recognizers.
- Finally, if end-users may use low-resource devices for interacting with your gesture-based UI, favour Penny Pincher and $Q, which are known to perform well with limited processing power and computational time.

4.2 Comparative Testing

The decision table reveals only the experimental conditions in which the recognizers have proven accurate. For example, Jackknife was tested with uni-strokes, but not with multi-strokes, and none can say whether or not it is also accurate with multi-strokes. Moreover, the heterogeneity of the experimental conditions under which the recognizers were tested does not allow them to be compared with each other on the same basis. Indeed, we cannot determine if another algorithm is more performant when one recognizer only is tested on a dataset. For this purpose, a comparative testing of all recognizers should be performed for evaluating their benefits and shortcomings based on predefined sets of datasets, criteria, and contextual variables (*e.g.*, how recognizers behave with other stroke counts and modalities (*i.e.*, uni- and multi-strokes from mouse, pen, and finger), other touch-sensitive devices (*i.e.*, smartphones, smartwatches), and in both user-dependent and user-independent scenarios (*e.g.*, one can evaluate $1 in a user-independent scenario to investigate its accuracy in this condition)).

4.3 Evaluation Scenarios

Researchers and practitioners aspire to assess recognition accuracy in additional contexts of use. The $1 and $N-MMG datasets include, for each gesture class and each participant, samples performed at three different speeds, *i.e.*, slow, medium, and fast, opening up the possibility to evaluate the impact of speed on recognition accuracy, *i.e.*, to assess if stroke recognizers are speed-(in)dependent. Vatavu et al. [75] investigate the impact of low-vision on touch gesture articulation, collecting gestures from users with *and* without visual impairments (*e.g.*, blurred vision, blind spots). This particular dataset enables evaluating the impact of low-vision on recognition

accuracy. Ultimately, all datasets including two or more distinct criteria during gesture collection are subject to extend the evaluation scenarios and investigate the impact of these criteria on recognition accuracy. For example, researchers and practitioners can assess the influence of some input device (*e.g.* smartphone *vs.* tablet), speed (*e.g.*, slow, medium, fast), user type (*e.g.* low-vision *vs.* full-vision users), and modality (*e.g.*, mouse, pen, finger) based on existing and new datasets acquired for the purpose. In these conditions, a first criteria-independent evaluation would consist of training with gestures performed according to a first criteria (*e.g.*, with a smartphone), and testing with gestures performed according to a second criteria (*e.g.*, with a tablet). A second criteria-independent evaluation would consist of training with gestures performed according to a first criteria, and testing with gestures performed according to two or more independent criteria (*e.g.*, with a smartphone, a tablet, and a smartwatch).

4.4 User-Defined Gesture Sets

Most devices or sensors are shipped with their own set of *system-defined gestures*. For instance, Logbar's Ring Zero device comes with a simple set of five system-defined gestures, such as swipes and checkmark. Similarly, some operating systems or toolkits impose their own set of system-defined gestures, which could be appreciated with various degrees by end-users. Designers with little or no experience in gesture-based UIs could also suggest *designer-defined gestures* that are challenging to classify by recognizers and/or difficult to remember by end-users. Involving the end-user in the process of creating both easy-to-discover and easy-to-learn gesture sets has been proved as positively impacting the overall usability [45]. For example, researchers have conducted a wide set of Gesture Elicitation Studies (GES) for many different types of modalities (*e.g.*, pen, finger, mid-air, and whole-body) and applications [6, 7, 13, 21, 23, 58, 70, 82, 83]. See [78] for a survey of GES. A GES prompts participants to suggest their own preferred gestures for carrying out a series of tasks through a gesture-based UI. The objective is to come up with a consensus set of user-defined gestures [72, 76, 77].

The recognizers analyzed in this survey were solely evaluated with gesture sets defined by UI experts, and not by end-users. Moreover, most recognizers promoted their own gesture set(s) for testing, which is prone to overfitting when the recognizer is so tied to the gesture set that it displays an exceptional recognition rate, but hardly transferable. The stroke recognition community now needs one or several common gesture sets to compare all recognizers in the same conditions: in terms of the number of participants, the nature, and the number of gesture classes, the number of samples per gesture class, the modalities, and the devices used for sampling the gestures.

4.5 Eager (On-the-Fly) Recognition

GRANDMA and G-Gene can be extended with eager recognition, *i.e.*, the ability to classify gestures as soon as they are unambiguous (on-the-fly). If the accuracy of on-the-fly recognition has never been evaluated for GRANDMA, G-Gene reached a recognition rate of 88.2% in an online condition, *i.e.*, when the gestures were replayed one timestamp at a time to simulate their production. There is still room for improving the classification of ongoing stroke gestures as the $P and the Protractor algorithms reached 96.6% in an offline condition for the same dataset. This will be also useful for feedforward and guidance. For instance, OctoPocus [9] displays partial strokes of a gesture being recognized on-the-fly.

4.6 Guidance

Beyond de facto standard gestures, such as flicks, swipes, pinch-in/out that are widespread, advanced gesture-based UIs require a broader vocabulary which may be difficult to "reveal" to novice end-users. Uncommon or unfamiliar gestures invite engineering a guidance system helping end-users in discovering, performing, and remembering gestures. For this purpose, G-Gene combined the recognition accuracy offered by HMMs and the guidance provided during gesture performance. Full guidance should explicitly address three sub-properties: (1)

feedback to help the end user in understanding which part of the gesture has already been issued, (2) *feedup* to determine which sub-stroke is being subject to recognition, and (3) *feedforward* to help the end user in discovering, exploring, and selecting the rest of the gesture to trigger the command. Unfortunately, few of the analyzed recognizers support feedback and/or feedforward, and the transition from novice to expert end-users.

4.7 On-demand ARST-invariance

Significant variations in articulation, rotation, scale, and position appear from one context of use to another, due to the variation of end-users (*e.g.*, participants who trained the recognizer are rarely the end-users of the gesture-based UI), input devices (*e.g.*, input device, sensor, or platform used at run-time is rarely the same as the one used for training), and physical environments (*e.g.*, people issue candidate gestures in the real world as opposed to the training samples performed in a controlled environment). Most recognizers are scaling- and translation-invariant, yet do not satisfy articulation- and rotation-invariance. !FTL is the first recognizer achieving naturally rotation-invariance in constrained time, whereas its predecessors used time-consuming techniques to achieve it, such as the Golden Section Search. The recognizer still fails to satisfy articulation-invariance and does not support any flexibility in enabling or disabling some of its invariances. There are therefore motivations for going beyond simple RST-invariance and introducing an ARST-invariant multi-stroke recognizer. Even better, the invariance properties should be controllable on-demand since the context of use could impose or relax constraints only determined at run-time. For example, interacting around a circular tabletop imposes rotation-invariance while interacting on a trackpad relaxes it. Usually, recognizers are scaling-invariant, but the scale constraint becomes undesired if the gesture scale is tightly coupled to a gesture parameter. For example, issuing a "$>$" gesture assigned to "Move forward" in a media player could be regulated by its size: the larger the gesture is, the faster the move forward progresses, instead of repeating the gesture multiple times.

5 CONCLUSION

We conducted a targeted literature review to identify 16 significant 2D stroke gesture recognizers. In a descriptive analysis, we presented their motivations, we explained their algorithm in terms of preprocessing, gesture representation, and classification, we discussed their invariance properties, and reported their performance. In a comparative analysis, we have discussed their similarities and differences to generate ideas and future avenues in 2D stroke gesture recognition. First, we suggested a decision table for UI prototypers, designers, and researchers who wish to choose the right recognizer given a gesture set and a context of use. Second, we highlighted room for improvements in six distinct areas: (1) perform a comparative testing of all recognizers with predefined datasets, criteria, and contextual variables, (2) define and collect a user-elicited gesture set to be added in this comparative testing, (3) new scenarios of evaluation to match real use-cases, (4) eager recognition for classifying gestures as they are performed by end-users, (5) guidance systems to enhance the usability and the discoverability of gesture-based UIs, and (6) on-demand ARST-invariance to flexibly control gesture production and classification.

ACKNOWLEDGMENTS

The work is supported by the UCLouvain Fonds Speciaux de Recherche (FSR) under grant number 61273304 and by FNRS Fonds pour la Formation à la Recherche dans l'Industrie et l'Agriculture (FRIA) under convention number FC 31773. The authors would like to thank the Louvain Research Institute in Management and Organizations (LouRIM) for its continuous support too.

REFERENCES

[1] 2001. LibStroke - Stroke Translation Library. http://www.gnu.msn.by/directory/gui/other/libstroke.html. Accessed: 2019-12-17.

[2] Ahmed Kadem Hamed Al-Saedi and Abbas H Hassin Al-Asadi. 2019. Survey of Hand Gesture Recognition Systems. *Journal of Physics: Conference Series* 1294 (sep 2019), 042003. https://doi.org/10.1088/1742-6596/1294/4/042003

[3] Lisa Anthony and Jacob O. Wobbrock. 2010. A Lightweight Multistroke Recognizer for User Interface Prototypes. In *Proceedings of Graphics Interface 2010 (GI '10)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 245–252. http://dl.acm.org/citation.cfm?id=1839214.1839258

[4] Lisa Anthony and Jacob O. Wobbrock. 2012. $N-protractor: A Fast and Accurate Multistroke Recognizer. In *Proceedings of Graphics Interface 2012 (GI '12)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 117–120. http://dl.acm.org/citation.cfm?id=2305276.2305296

[5] Caroline Appert and Shumin Zhai. 2009. Using Strokes as Command Shortcuts: Cognitive Benefits and Toolkit Support. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. Association for Computing Machinery, New York, NY, USA, 2289–2298. https://doi.org/10.1145/1518701.1519052

[6] Shaikh Shawon Arefin Shimon, Courtney Lutton, Zichun Xu, Sarah Morrison-Smith, Christina Boucher, and Jaime Ruiz. 2016. Exploring Non-Touchscreen Gestures for Smartwatches. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3822–3833. https://doi.org/10.1145/2858036.2858385

[7] Ilhan Aslan, Tabea Schmidt, Jens Woehrle, Lukas Vogel, and Elisabeth André. 2018. Pen + Mid-Air Gestures: Eliciting Contextual Gestures. In *Proceedings of the 20th ACM International Conference on Multimodal Interaction (ICMI '18)*. Association for Computing Machinery, New York, NY, USA, 135–144. https://doi.org/10.1145/3242969.3242979

[8] Ruben Balcazar, Francisco R. Ortega, Katherine Tarre, Armando Barreto, Mark Weiss, and Naphtali D. Rishe. 2017. CircGR: Interactive Multi-Touch Gesture Recognition Using Circular Measurements. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces (ISS '17)*. Association for Computing Machinery, New York, NY, USA, 12–21. https://doi.org/10.1145/3132272.3134139

[9] Olivier Bau and Wendy E. Mackay. 2008. OctoPocus: A Dynamic Guide for Learning Gesture-Based Command Sets. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST '08)*. Association for Computing Machinery, New York, NY, USA, 37–46. https://doi.org/10.1145/1449715.1449724

[10] Xiaojun Bi, Ciprian Chelba, Tom Ouyang, Kurt Partridge, and Shumin Zhai. 2012. Bimanual Gesture Keyboard. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. Association for Computing Machinery, New York, NY, USA, 137–146. https://doi.org/10.1145/2380116.2380136

[11] K. K. Biswas and S. K. Basu. 2011. Gesture recognition using Microsoft Kinect. In *The 5th International Conference on Automation, Robotics and Applications*. 100–103.

[12] Rachel Blagojevic, Samuel Hsiao-Heng Chang, and Beryl Plimmer. 2010. The Power of Automatic Feature Selection: Rubine on Steroids. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium (SBIM '10)*. Eurographics Association, Goslar, DEU, 79–86.

[13] Idil Bostan, Oundefineduz Turan Buruk, Mert Canat, Mustafa Ozan Tezcan, Celalettin Yurdakul, Tilbe Göksun, and Oundefineduzhan Özcan. 2017. Hands as a Controller: User Preferences for Hand Specific On-Skin Gestures. In *Proceedings of the 2017 Conference on Designing Interactive Systems (DIS '17)*. Association for Computing Machinery, New York, NY, USA, 1123–1134. https://doi.org/10.1145/3064663.3064766

[14] Alessandro Carcangiu and Lucio Davide Spano. 2018. G-Gene: A Gene Alignment Method for Online Partial Stroke Gestures Recognition. *Proc. ACM Hum.-Comput. Interact.* 2, EICS, Article 13 (June 2018), 17 pages. https://doi.org/10.1145/3229095

[15] Alessandro Carcangiu, Lucio Davide Spano, Giorgio Fumera, and Fabio Roli. 2019. DEICTIC: A compositional and declarative gesture description based on hidden markov models. *International Journal of Human-Computer Studies* 122 (2019), 113–132. https://doi.org/10.1016/j.ijhcs.2018.09.001

[16] Xiang "Anthony" Chen, Julia Schwarz, Chris Harrison, Jennifer Mankoff, and Scott E. Hudson. 2014. Air+touch: Interweaving Touch & in-Air Gestures. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. Association for Computing Machinery, New York, NY, USA, 519–525. https://doi.org/10.1145/2642918.2647392

[17] Hong Cheng, Lu Yang, and Zicheng Liu. 2016. Survey on 3D Hand Gesture Recognition. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 9 (2016), 1659–1673. https://doi.org/10.1109/TCSVT.2015.2469551

[18] Adrien Coyette, Sascha Schimke, Jean Vanderdonckt, and Claus Vielhauer. 2007. Trainable Sketch Recognizer for Graphical User Interface Design. In *Human-Computer Interaction – INTERACT 2007*, Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 124–135.

[19] Stefano Dessì and Lucio Davide Spano. 2020. DG3: Exploiting Gesture Declarative Models for Sample Generation and Online Recognition. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 82 (June 2020), 21 pages. https://doi.org/10.1145/3397870

[20] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. *Proc. VLDB Endow.* 1, 2 (Aug. 2008), 1542–1552. https://doi.org/10.14778/1454159.1454226

[21] Tilman Dingler, Rufat Rzayev, Alireza Sahami Shirazi, and Niels Henze. 2018. Designing Consistent Gestures Across Device Types: Eliciting RSVP Controls for Phone, Watch, and Glasses. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, Article Paper 419, 12 pages. https://doi.org/10.1145/3173574.3173993

[22] Richard O. Duda and Peter E. Hart. 1974. Pattern Classification and Scene Analysis. *The Library Quarterly* 44, 3 (1974), 258–259. https://doi.org/10.1086/620282 arXiv:https://doi.org/10.1086/620282

[23] Bogdan-Florin Gheran, Jean Vanderdonckt, and Radu-Daniel Vatavu. 2018. Gestures for Smart Rings: Empirical Results, Insights, and Design Implications. In *Proceedings of the 2018 Designing Interactive Systems Conference (DIS '18)*. Association for Computing Machinery, New York, NY, USA, 623–635. https://doi.org/10.1145/3196709.3196741

[24] John D. Gould and Josiane Salaun. 1986. Behavioral Experiments on Handmarkings. *SIGCHI Bull.* 18, 4 (May 1986), 175–181. https://doi.org/10.1145/1165387.275626

[25] J. Herold and T. F. Stahovich. 2012. The 1Cent 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, Goslar Germany, Germany, 39–46. http://dl.acm.org/citation.cfm?id=2331067.2331074

[26] Heloise Hse, Michael Shilman, and A. Richard Newton. 2004. Robust Sketched Symbol Fragmentation Using Templates. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04)*. Association for Computing Machinery, New York, NY, USA, 156–160. https://doi.org/10.1145/964442.964472

[27] Yoonho Hwang. 2012. A Fast Nearest Neighbor Search Algorithm by Nonlinear Embedding. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (CVPR '12)*. IEEE Computer Society, Washington, DC, USA, 3053–3060. http://dl.acm.org/citation.cfm?id=2354409.2354783

[28] Yoonho Hwang and Hee-Kap Ahn. 2011. Convergent Bounds on the Euclidean Distance. In *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS'11)*. Curran Associates Inc., USA, 388–396. http://dl.acm.org/citation.cfm?id=2986459.2986503

[29] Noor Adnan Ibraheem and Rafiqul Zaman Khan. 2012. Survey on Various Gesture Recognition Technologies and Techniques. *International Journal of Computer Applications* 50, 7 (7 2012), 38–44. https://doi.org/10.5120/7786-0883

[30] Rafiqul Zaman Khan and Noor Adnan Ibraheem. 2012. Survey on Gesture Recognition for Hand Image Postures. *Comput. Inf. Sci.* 5, 3 (2012), 110–121. https://doi.org/10.5539/cis.v5n3p110

[31] Kenrick Kin, Tom Miller, Björn Bollensdorff, Tony DeRose, Björn Hartmann, and Maneesh Agrawala. 2011. Eden: A Professional Multitouch Tool for Constructing Virtual Organic Environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. Association for Computing Machinery, New York, NY, USA, 1343–1352. https://doi.org/10.1145/1978942.1979141

[32] Barbara Kitchenham, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. 2010. Systematic literature reviews in software engineering – A tertiary study. *Information and Software Technology* 52, 8 (2010), 792–805. https://doi.org/10.1016/j.infsof.2010.03.006

[33] Sven Kratz and Michael Rohs. 2010. The $3 Recognizer: Simple 3D Gesture Recognition on Mobile Devices. In *Proceedings of the 15th International Conference on Intelligent User Interfaces (IUI '10)*. Association for Computing Machinery, New York, NY, USA, 419–420. https://doi.org/10.1145/1719970.1720051

[34] Sven Kratz and Michael Rohs. 2011. Protractor3D: A Closed-Form Solution to Rotation-Invariant 3D Gestures. In *Proceedings of the 16th International Conference on Intelligent User Interfaces (IUI '11)*. Association for Computing Machinery, New York, NY, USA, 371–374. https://doi.org/10.1145/1943403.1943468

[35] Per-Ola Kristensson and Shumin Zhai. 2004. SHARK2: 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 '04)*. ACM, New York, NY, USA, 43–52. https://doi.org/10.1145/1029632.1029640

[36] Gordon Kurtenbach, George Fitzmaurice, Thomas Baudel, and Bill Buxton. 1997. The Design of a GUI Paradigm Based on Tablets, Two-hands, and Transparency. In *Proceedings of the ACM International Conference on Human Factors in Computing Systems (CHI '97)*. ACM, New York, NY, USA, 35–42. https://doi.org/10.1145/258549.258574

[37] Lynn Kysh. 2013. Difference between a systematic review and a literature review. (8 2013). https://doi.org/10.6084/m9.figshare.766364.v1

[38] Joseph J. LaViola. 2013. 3D Gestural Interaction: The State of the Field. *International Scholarly Research Notices* 2013, Article 514641 (2013), 18 pages. https://doi.org/10.1155/2013/514641

[39] Luis A. Leiva, Daniel Martín-Albo, and Réjean Plamondon. 2015. Gestures à Go Go: Authoring Synthetic Human-Like Stroke Gestures Using the Kinematic Theory of Rapid Movements. *ACM Trans. Intell. Syst. Technol.* 7, 2, Article 15 (Nov. 2015), 29 pages. https://doi.org/10.1145/2799648

[40] Yang Li. 2010. 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, 2169–2172. https://doi.org/10.1145/1753326.1753654

[41] L. M. Lorigo and V. Govindaraju. 2006. Offline Arabic handwriting recognition: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 5 (May 2006), 712–724. https://doi.org/10.1109/TPAMI.2006.102

[42] Roanna Lun and Wenbing Zhao. 2015. A Survey of Applications and Human Motion Recognition with Microsoft Kinect. *Int. J. Pattern Recognit. Artif. Intell.* 29, 5 (2015), 1555008:1–1555008:48. https://doi.org/10.1142/S0218001415550083

[43] Lorenzo Luzzi and Paolo Roselli. 2020. The shape of planar smooth gestures and the convergence of a gesture recognizer. *Aequationes mathematicae* (2020). https://doi.org/10.1007/s00010-020-00712-7

[44] N. Magrofuoco, J. Pérez-Medina, P. Roselli, J. Vanderdonckt, and S. Villarreal. 2019. Eliciting Contact-Based and Contactless Gestures With Radar-Based Sensors. *IEEE Access* 7 (2019), 176982–176997.

[45] Miguel A. Nacenta, Yemliha Kamber, Yizhou Qiang, and Per Ola Kristensson. 2013. Memorability of Pre-Designed and User-Defined Gesture Sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 1099–1108. https://doi.org/10.1145/2470654.2466142

[46] Christos H. Papadimitriou and Kenneth Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[47] Brandon Paulson and Tracy Hammond. 2008. PaleoSketch: Accurate Primitive Sketch Recognition and Beautification. In *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI '08)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/1378773.1378775

[48] Corey Pittman, Eugene M. Taranta II, and Joseph J. LaViola. 2016. A $-Family Friendly Approach to Prototype Selection. In *Proceedings of the 21st International Conference on Intelligent User Interfaces (IUI '16)*. Association for Computing Machinery, New York, NY, USA, 370–374. https://doi.org/10.1145/2856767.2856808

[49] R. Plamondon and S. N. Srihari. 2000. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 1 (Jan 2000), 63–84. https://doi.org/10.1109/34.824821

[50] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 1992. *Numerical Recipes in C* (second ed.). Cambridge University Press, Cambridge, USA.

[51] Francis Quek, David McNeill, Robert Bryll, Susan Duncan, Xin-Feng Ma, Cemil Kirbas, Karl E. McCullough, and Rashid Ansari. 2002. Multimodal Human Discourse: Gesture and Speech. *ACM Trans. Comput.-Hum. Interact.* 9, 3 (Sept. 2002), 171–193. https://doi.org/10.1145/568513.568514

[52] Siddharth S. Rautaray and Anupam Agrawal. 2015. Vision based hand gesture recognition for human computer interaction: a survey. *Artif. Intell. Rev.* 43, 1 (2015), 1–54. https://doi.org/10.1007/s10462-012-9356-9

[53] J. Reaver, T. F. Stahovich, and J. Herold. 2011. How to Make a Quick$: Using Hierarchical Clustering to Improve the Efficiency of the Dollar Recognizer. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling (SBIM '11)*. ACM, New York, NY, USA, 103–108. https://doi.org/10.1145/2021164.2021183

[54] Yosra Rekik, Radu-Daniel Vatavu, and Laurent Grisoni. 2014. Match-up & Conquer: A Two-Step Technique for Recognizing Unconstrained Bimanual and Multi-Finger Touch Input. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces (AVI '14)*. Association for Computing Machinery, New York, NY, USA, 201–208. https://doi.org/10.1145/2598153.2598167

[55] J.R. Rhyne and C.G. Wolf. 1986. *Gestural Interfaces for Information Processing Applications*. International Business Machines Incorporated, Thomas J. Watson Research Center. https://books.google.be/books?id=NzVsGwAACAAJ

[56] Dean Rubine. 1991. Specifying Gestures by Example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. ACM, New York, NY, USA, 329–337. https://doi.org/10.1145/122718.122753

[57] Stefan Schneegass and Alexandra Voit. 2016. GestureSleeve: Using Touch Sensitive Fabrics for Gestural Input on the Forearm for Controlling Smartwatches. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers (ISWC '16)*. Association for Computing Machinery, New York, NY, USA, 108–115. https://doi.org/10.1145/2971763.2971797

[58] Adwait Sharma, Joan Sol Roo, and Jürgen Steimle. 2019. Grasping Microgestures: Eliciting Single-Hand Microgestures for Handheld Objects. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article Paper 402, 13 pages. https://doi.org/10.1145/3290605.3300632

[59] Jia Sheng. 2003. *A Study of AdaBoost in 3D Gesture Recognition*. Technical Report. Computer Science Department, Toronto University, Toronto, ON. http://www.dgp.toronto.edu/~jsheng/doc/CSC2515/Report.pdf

[60] C. Y. Suen, M. Berthod, and S. Mori. 1980. Automatic recognition of handprinted characters—The state of the art. *Proc. IEEE* 68, 4 (April 1980), 469–487. https://doi.org/10.1109/PROC.1980.11675

[61] C. C. Tappert. 1982. Cursive Script Recognition by Elastic Matching. *IBM J. Res. Dev.* 26, 6 (Nov. 1982), 765–771. https://doi.org/10.1147/rd.266.0765

[62] Eugene M. Taranta, II and Joseph J. LaViola, Jr. 2015. Penny Pincher: A Blazing Fast, Highly Accurate $-family Recognizer. In *Proceedings of the 41st Graphics Interface Conference (GI '15)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 195–202. http://dl.acm.org/citation.cfm?id=2788890.2788925

[63] Eugene M. Taranta, II, Mehran Maghoumi, Corey R. Pittman, and Joseph J. LaViola, Jr. 2016. A Rapid Prototyping Approach to Synthetic Data Generation for Improved 2D Gesture Recognition. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 873–885. https://doi.org/10.1145/2984511.2984525

[64] Eugene M. Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R. Pittman, and Joseph J. LaViola Jr. 2017. Jackknife: A Reliable Recognizer with Few Samples and Many Modalities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 5850–5861. https://doi.org/10.1145/3025453.3026002

[65] Seiichi Uchida and Hiroaki Sakoe. 2005. A Survey of Elastic Matching Techniques for Handwritten Character Recognition. *IEICE - Trans. Inf. Syst.* E88-D, 8 (Aug. 2005), 1781–1790. https://doi.org/10.1093/ietisy/e88-d.8.1781

[66] Jean Vanderdonckt, Bruno Dumas, and Mauro Cherubini. 2018. Comparing Some Distances in Template-Based 2D Gesture Recognition. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems (CHI EA '18)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3170427.3188452

[67] Jean Vanderdonckt, Paolo Roselli, and Jorge Luis Pérez-Medina. 2018. !FTL, an Articulation-Invariant Stroke Gesture Recognizer with Controllable Position, Scale, and Rotation Invariances. In *Proceedings of the 20th ACM International Conference on Multimodal Interaction (ICMI '18)*. Association for Computing Machinery, New York, NY, USA, 125–134. https://doi.org/10.1145/3242969.3243032

[68] Radu-Daniel Vatavu. 2013. The impact of motion dimensionality and bit cardinality on the design of 3D gesture recognizers. *Int. J. Hum. Comput. Stud.* 71, 4 (2013), 387–409. https://doi.org/10.1016/j.ijhcs.2012.11.005

[69] Radu-Daniel Vatavu. 2012. 1F: One Accessory Feature Design for Gesture Recognizers. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces (IUI '12)*. ACM, New York, NY, USA, 297–300. https://doi.org/10.1145/2166966.2167022

[70] Radu-Daniel Vatavu. 2012. User-Defined Gestures for Free-Hand TV Control. In *Proceedings of the 10th European Conference on Interactive TV and Video (EuroITV '12)*. Association for Computing Machinery, New York, NY, USA, 45–48. https://doi.org/10.1145/2325616.2325626

[71] Radu-Daniel Vatavu. 2017. Improving Gesture Recognition Accuracy on Touch Screens for Users with Low Vision. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4667–4679. https://doi.org/10.1145/3025453.3025941

[72] Radu-Daniel Vatavu. 2019. The Dissimilarity-Consensus Approach to Agreement Analysis in Gesture Elicitation Studies. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article Paper 224, 13 pages. https://doi.org/10.1145/3290605.3300454

[73] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2012. 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, 273–280. https://doi.org/10.1145/2388676.2388732

[74] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2018. $Q: A Super-quick, Articulation-invariant Stroke-gesture Recognizer for Low-resource Devices. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '18)*. ACM, New York, NY, USA, Article 23, 12 pages. https://doi.org/10.1145/3229434.3229465

[75] Radu-Daniel Vatavu, Bogdan Gheran, and Maria-Doina Schipor. 2018. The Impact of Low Vision on Touch-Gesture Articulation on Mobile Devices. *IEEE Pervasive Computing* 17, 1 (2018), 27–37. https://doi.org/10.1109/MPRV.2018.011591059

[76] Radu-Daniel Vatavu and Jacob O. Wobbrock. 2015. Formalizing Agreement Analysis for Elicitation Studies: New Measures, Significance Test, and Toolkit. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 1325–1334. https://doi.org/10.1145/2702123.2702223

[77] Radu-Daniel Vatavu and Jacob O. Wobbrock. 2016. Between-Subjects Elicitation Studies: Formalization and Tool Support. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3390–3402. https://doi.org/10.1145/2858036.2858228

[78] Santiago Villarreal-Narvaez, Jean Vanderdonckt, Radu-Daniel Vatavu, and Jacob O. Wobbrock. 2020. A Systematic Review of Gesture Elicitation Studies: What Can We Learn from 216 Studies?. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference (DIS '20)*. Association for Computing Machinery, New York, NY, USA, 855–872. https://doi.org/10.1145/3357236.3395511

[79] Petra Wagner, Zofia Malisz, and Stefan Kopp. 2014. Guest Editorial: Gesture and Speech in Interaction: An Overview. *Speech Commun.* 57 (Feb. 2014), 209–232. https://doi.org/10.1016/j.specom.2013.09.008

[80] Cheng-Yao Wang, Min-Chieh Hsiu, Po-Tsung Chiu, Chiao-Hui Chang, Liwei Chan, Bing-Yu Chen, and Mike Y. Chen. 2015. PalmGesture: Using Palms as Gesture Interfaces for Eyes-Free Input. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '15)*. Association for Computing Machinery, New York, NY, USA, 217–226. https://doi.org/10.1145/2785830.2785885

[81] Don Willems, Ralph Niels, Marcel van Gerven, and Louis Vuurpijl. 2009. Iconic and Multi-Stroke Gesture Recognition. *Pattern Recogn.* 42, 12 (Dec. 2009), 3303–3312. https://doi.org/10.1016/j.patcog.2009.01.030

[82] Markus L. Wittorf and Mikkel R. Jakobsen. 2016. Eliciting Mid-Air Gestures for Wall-Display Interaction. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction (NordiCHI '16)*. Association for Computing Machinery, New York, NY, USA, Article Article 3, 4 pages. https://doi.org/10.1145/2971485.2971503

[83] Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. 2009. User-Defined Gestures for Surface Computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1083–1092. https://doi.org/10.1145/1518701.1518866

[84] Jacob O. Wobbrock and Brad A. Myers. 2006. From Letters to Words: Efficient Stroke-Based Word Completion for Trackball Text Entry. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility (Assets '06)*. Association for Computing Machinery, New York, NY, USA, 2–9. https://doi.org/10.1145/1168987.1168990

[85] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. 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, 159–168. https://doi.org/10.1145/1294211.1294238

[86] Carl D. Worth. 2003. xstroke: Full-screen Gesture Recognition for X. In *Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference, June 9-14, 2003, San Antonio, Texas, USA*. 187–196. http://www.usenix.org/events/usenix03/tech/freenix03/worth.html

[87] Robert Xiao, Gierad Laput, and Chris Harrison. 2014. Expanding the Input Expressivity of Smartwatches with Mechanical Pan, Twist, Tilt and Click. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. Association for Computing Machinery, New York, NY, USA, 193–196. https://doi.org/10.1145/2556288.2557017

[88] Yina Ye and Petteri Nurmi. 2015. Gestimator: Shape and Stroke Similarity Based Gesture Recognition. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction (ICMI '15)*. Association for Computing Machinery, New York, NY, USA, 219–226. https://doi.org/10.1145/2818346.2820734

[89] Shumin Zhai and Per-Ola Kristensson. 2003. Shorthand Writing on Stylus Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03)*. ACM, New York, NY, USA, 97–104. https://doi.org/10.1145/642611.642630

[90] S. Zhai, P. O. Kristensson, C. Appert, T. H. Andersen, and X. Cao. 2012. *Foundational Issues in Touch-Surface Stroke Gesture Design: An Integrative Review*. now. https://ieeexplore.ieee.org/document/8187096

[91] Yougen Zhang, Wei Deng, Hanchen Song, and Lingda Wu. 2013. A Fast Pen Gesture Matching Method Based on Nonlinear Embedding. In *Advances in Image and Graphics Technologies*, Tieniu Tan, Qiuqi Ruan, Xilin Chen, Huimin Ma, and Liang Wang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 223–231.

[92] Jingjie Zheng and Daniel Vogel. 2016. Finger-Aware Shortcuts. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 4274–4285. https://doi.org/10.1145/2858036.2858355