μV: An Articulation, Rotation, Scaling, and Translation Invariant (ARST) Multi-stroke Gesture Recognizer

NATHAN MAGROFUOCO, Université catholique de Louvain, LouRIM, Belgium
PAOLO ROSELLI, Università degli Studi di Roma "Tor Vergata", Roma, Italy, Italy
JEAN VANDERDONCKT, Université catholique de Louvain, LouRIM & ICTeam, Belgium

Finger-based gesture input becomes a major interaction modality for surface computing. Due to the low precision of the finger and the variation in gesture production, multistroke gestures are still challenging to recognize in various setups. In this paper, we present μV, a multistroke gesture recognizer that addresses the properties of articulation, rotation, scaling, and translation invariance by combining $P+'s cloud-matching for articulation invariance with !FTL's local shape distance for RST-invariance. We evaluate μV against five competitive recognizers on MMG, an existing gesture set, and on two new versions for smartphones and tablets, MMG+ and RMMG+, a randomly rotated version on both platforms. μV is significantly more accurate than its predecessors when rotation invariance is required and not significantly inferior when it is not. μV is also significantly faster than others with many samples and not significantly slower with few samples.

CCS Concepts: • **Human-centered computing** → **Graphical user interfaces**; **Gestural input**.

Additional Key Words and Phrases: Gesture recognition, multistroke gestures, invariance properties

ACM Reference Format:
Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2022. μV: An Articulation, Rotation, Scaling, and Translation Invariant (ARST) Multi-stroke Gesture Recognizer. *Proc. ACM Hum.-Comput. Interact.* 6, EICS, Article 150 (June 2022), 25 pages. https://doi.org/10.1145/3532200

1 INTRODUCTION

The multiplication of pen- and touch-sensitive flat surfaces [24] increasingly present opportunities for using 2-dimensional (2D) pen and finger gestures in User Interfaces (UIs) for surface computing [43, 44]. Gesture-based interaction offers an intuitive alternative to the traditional keyboard, menu, and direct manipulation interfaces. The use of touch gestures has become ubiquitous on mobile devices; "pinching" to zoom on a picture and "swiping" for navigating contents are now gestures belonging to the consensual vocabulary and practice. The popularization of gesture-based interaction constantly raises the need to associate new gestures with additional functions. However, most mobile operating systems support novel gestures with the pen and/or finger in a programmatic way.

Authors' addresses: Nathan Magrofuoco, nathan.magrofuoco@uclouvain.be, Université catholique de Louvain, LouRIM, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium; Paolo Roselli, Università degli Studi di Roma "Tor Vergata", Roma, Italy, Piazzale Aldo Moro, 5, Roma, 00185, Italy, roselli@mat.uniroma2.it; Jean Vanderdonckt, jean.vanderdonckt@uclouvain.be, Université catholique de Louvain, LouRIM & ICTeam, Place des Doyens, 1 & Place du Levant, 3, Louvain-la-Neuve, 1348, Belgium, jean.vanderdonckt@uclouvain.be.

2573-0142/2022/6-ART150 $15.00
https://doi.org/10.1145/3532200

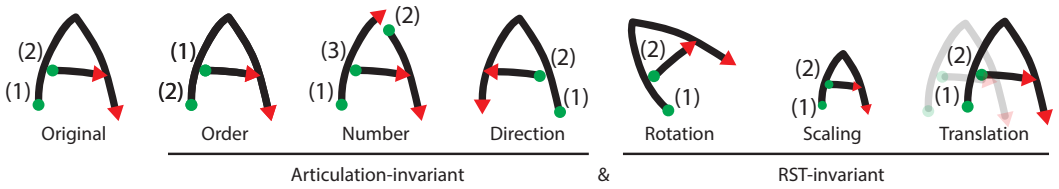

Fig. 1. Stroke gestures can be articulated in multiple ways owed to the order, number, and direction in which the strokes can be drawn (Articulation-invariance). Stroke gestures can be performed at different rotations, scales, and/or translations (RST-invariance); hence the need for an ARST-invariant multistroke recognizer.

Several simple, fast, and accurate recognizers are today available for classifying easy-to-remember and easy-to-reproduce gestures and for incorporating them into mainstream user interface development. These *stroke gestures* are hereby referred to as any movement trajectories captured with a stylus, a finger, or a mouse on a flat surface, and used as shortcuts to indicate scope and commands [4, 21, 22, 46, 47]. Movement trajectories, or *paths*, commonly represent letters, digits, geometric shapes, lines, symbols, and other patterns. Stroke gestures are usually acquired through a touch-sensitive surface (*e.g.* for sketching [23]), sampled, and then encoded as time-ordered sequences of (x, y) points [1].

Historically, the Rotation-Scaling-Translation (RST) paradigm [8] became a *de facto* standard to characterize flat stroke gestures: from two (or more) reference points captured, meaningful gesture parameters, also called *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. Invariance with respect to rotation, scaling, and translation is crucial 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 (unistroke *vs.* multistroke), and in which direction [7].

To address these invariance properties, Rubine implemented a feature-based classifier [22] to recognize unistroke gestures performed with a mouse. The $-family [1, 36, 37, 39, 44] introduced a set of K-nearest neighbor (k=1) template matchers to recognize uni-stroke, then multi-stroke gestures. Vector-based recognizers, such as PennyPincher [29], Jackknife [30], and !FTL [33], also demonstrated stroke recognition efficiency in various scenarios.

These recognizers, among others, satisfy different sets of invariance properties under varying circumstances. For example, the $1 algorithm [44] satisfies RST-invariance for uni-stroke gestures, while the $N algorithm [1] additionally satisfies articulation invariance: to accurately classify multistroke gestures, the order [41], the number, and direction in which the strokes are produced can be ignored. The $P [37], $P+ [36] and $Q algorithms [39] implement cloud matching to fasten the classification of multi-strokes; they consider gestures as unordered sets of points. These quick, highly accurate recognizers have dropped support for rotation invariance as it requires comparing two gestures at different angular adjustments, which is a resource- and time-consuming task.

Fig. 1 illustrates the need for rotation invariance with six different productions of the uppercase letter "A". Uncorrected slant may lead to false recognition, as many strokes differ only in orientation. Hence, the importance for multi-stroke recognizers to remain insensitive to rotation, as well as Articulation, Scaling, and Translation, which we combine under the umbrella "ARST-invariant".

Practitioners might consider again adopting rotation invariance for several reasons. Rotation invariance is the last property to reach full spatial invariance, which provides a good tolerance to noise, variations, and articulations, which is necessary in two cases: (1) when no particular orientation is relevant for the end-users, such as in space or during scuba diving (orientation is not always relevant), and (2) when no particular orientation is relevant for the input device, such

as with tablets, tabletops, and round tables around which end-users issue gestures oriented in many ways. Our human motor system inevitably introduces variations in gesture articulation, even when trained, that arise from rotation invariance. Moreover, a significant amount of variations in gesture orientation already exists for one user, but also from one user to another, from one platform to another, and from one context of use to another [36]. For example, one hand holds the mobile device while the other produces gestures on the touch-sensitive surface, yet mobile devices are also held on a flat surface or used on the fly. Variations in posture, in the physical arrangement of the user's arms and shoulders, and in the handling of the devices lead to various slants during gesture production [45]. For interaction contexts with dynamic contents, variation in orientation has an important effect on the accuracy of today's recognizers [4, 28]. In addition, some datasets contain identical gestures with different angles. For example, NicIcon [42] requires distinguishing a lying body from a standing body for two different actions. There are also some specific designer-defined cases for which rotation-invariance is required, such as authentication gestures [25] and TouchTokens [3], which are captured by three points everywhere on the surface to recognize the corresponding gesture, thus being essential for consistency in map navigation. PalmGesture [40] also assumes that gestures can be performed in any orientation.

We contribute to multistroke gesture recognition as follows:

(1) We acknowledge the needs for combining articulation invariant and RST-invariant recognizers to create an ARST-invariant multistroke gesture recognizer.
(2) We implement μV, a multi-stroke recognizer satisfying ARST-invariance by combining \$P+'s cloud-matching for satisfying articulation invariance and the "local shape distance" introduced in the !FTL algorithm for RST-invariance.
(3) We perform an experiment with one existing dataset, MMG [38], and two novel publicly-released datasets, named MMG+ and RMMG+, to evaluate the potential benefits of μV with finger gestures on a smartphone and a tablet.
(4) The experimental findings suggest that μV is significantly more accurate than its predecessors when rotation-invariant recognition is necessary, and not significantly different when it is not. The μV algorithm is significantly faster than its predecessors, except Penny Pincher [29].
(5) We share a repository with the software developed for this paper, all recognizers written in JavaScript, the new μV recognizer, and the datasets converted into a unified JSON format to foster reproducible research around gesture recognition and its incorporation into mainstream interactive applications.

2 RELATED WORK

Several algorithms recognize 2D stroke gestures [16] by comparing a *candidate gesture* against *template gestures*. Each template corresponds to an example labeled by a *gesture class*, and the set of templates is called the *training set* since it is used to train the recognizer in advance. The comparison is performed by calculating a (dis)similarity distance between the candidate gesture and any template. Such template-matching recognizers can be classified into three families depending on how this distance is computed: (i) between-points, (ii) vectors between-points, and (iii) vector between-vectors.

Between-Points Distance. Rubine (R) [22] pioneered the aforementioned field of stroke gesture recognition by introducing an automatic algorithm: after a simple preprocessing to eliminate jiggle, a 13-feature vector is preliminary extracted from each gesture template. The candidate gesture is classified as one of the possible gesture classes via a linear evaluation of its 13-feature vector over the template feature vectors. This method offers several benefits: high recognition accuracy for a low computational cost; manual opportunistic programming is no longer required; its understanding is

	R	$1	$N	$P	$P+	$Q	PP	JK	!FTL	μV
Multi-stroke	X	X	✓	✓	✓	✓	X	X	X	✓
Articulation	X	✓	✓	✓	✓	✓	X	X	✓	✓
Rotation	✓	✓	✓	X	X	X	X	X	✓	✓
Scaling	X	✓	✓	✓	✓	✓	✓	✓	✓	✓
Translation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Not Combinatorial	✓	✓	X	✓	✓	✓	✓	✓	✓	✓

Table 1. Invariance properties of stroke gesture recognizers.

simple and facilitated by a geometric interpretation. After some preprocessing, $1 [44] computes the Euclidean distance between pairs of points of any couple (template, candidate) to find the template minimizing this distance. Uni-stroke recognition accuracy is high for about 100 lines of code involving only basic geometric and trigonometric operations. The $N recognizer [1] extends $1 to multistroke gesture recognition: each gesture is interpreted as a unistroke by connecting the spaces between the strokes that are made away from the sensing surface, *i.e.*, by following the user's hand in the air. All unistroke permutations of the gesture are automatically computed, which may produce a combinatorial explosion in both memory and execution time, despite two speed optimizations: *e.g.*, a simple square can be performed in 442 possible ways when considering the order, number, and direction of its strokes. $P [37] overcomes this issue by considering gestures as clouds of points, which are unordered sets of points. Each point from the candidate cloud can be matched with any point from the template cloud that has not been matched yet. The Euclidean distance is then computed between pairs of matching points, although other distances can be also considered, but without any proved benefit insofar. $P+ [36] improves $P flexibility such that any point in the candidate cloud can be matched to *any* point in the template cloud. $Q [39] introduces several algorithmic revisions of $P: the classification of multistroke gestures is faster when a large set of templates is involved, which is particularly efficient for low-end devices. Many variants to the $-family have gained some popularity, such as Protractor [13], $N-Protractor [2], Quick$ [20], 1F [35], and 1¢ [5].

Vectors Between-Points Distance. Penny Pincher (PP) [29] converts the resampled gesture points into a series of between-points vectors and calculates the sum of the angles between their corresponding vectors. Jackknife (JK) [30] is a multitool to recognize gestures sampled from many modalities such as 2D touch and 3D hand or full-body gestures. The algorithm converts the resampled points into a series of unit-length direction vectors, then uses Dynamic Time Warping (DTW) [14] to match a candidate and a template direction vector. Correction factors are computed to inflate the score of dissimilar gestures, and synthetic gestures are generated for learning rejection criteria.

Vector Between-Vectors Distance. !FTL [33] converts resampled gesture points into "shapes", a pair of two vectors between three consecutive points. Given the shapes of a candidate and a template, the recognizer calculates the local shape distance (LSD) between consecutive pairs of shapes to estimate the dissimilarity between the candidate and the template. The local shape distance satisfies RST-invariance after the resampling of the gestures, avoiding preprocessing required by the between-points distance recognizers.

Invariance Properties. Rubine and $1 (resp. Penny Pincher, Jackknife, and !FTL) are intrinsically unistroke recognizers since they match series of points (resp. series of vectors) in their order of appearance. Consequently, they are sensitive to gesture articulation. The $N, $P, $P+, and $Q algorithms are unistroke *and* multistroke recognizers since they are insensitive to the articulation of the strokes in order, number, and direction: $N, because all unistroke permutations of a multistroke are generated, and the others because they represent gestures as unordered sets of points during

the matching process. The $-family is also invariant to scaling and translation as template gestures are preprocessed in three steps: (1) gestures are resampled to N equidistantly-spaced points, (2) are scaled to a reference square, and (3) are translated so that their centroid is at the origin $(0, 0)$. Penny Pincher and Jackknife are invariant to scale and translation since they resample gestures to N equidistant points, then compute a vector between-points normalized dissimilarity score.

Rubine is invariant to translation as it does not operate at the point level, but it is scaling variant since it relies on features involving the length of the gesture's bounding box. The $P, $P+, and $Q, Penny Pincher, and Jackknife recognizers are sensitive to rotation as they do not rotate the gestures over a set of angular adjustments during the matching process, unlike $1 and $N which implement the Golden Section Search (GSS) [19] technique to do so. !FTL satisfies RST-invariance for unistroke gestures only, and $N faces a combinatorial explosion in algorithmic complexity despite satisfying ARST-invariance. Table 1 compares the aforementioned recognizers with respect to their properties of invariance, thus highlighting the need for our ARST-invariant multistroke recognizer, μV, without explosion.

3 OVERVIEW OF THE $P+ ALGORITHM

As our μV recognizer is built on $P+, a brief summary of this recognizer is reminded. To compute its between-points Euclidean distance, $P+ pre-processes the templates and any candidate to "normalize" them in terms of sampling, scaling, and translation:

(1) The gestures are resampled to a fixed number of equidistant points to avoid any variation in sampling due to the hardware and software;
(2) The resampled points are non-uniformly scaled to preserve the aspect ratio of 1D gestures, and to remain insensitive to scaling;
(3) The scaled, resampled points are translated so that their centroid is at the origin $(0, 0)$ to remain invariant to translation.

The $P+ algorithm compares any candidate against each stored template by cloud matching since the gestures are stored as unordered sets of points called *cloud*: any point from the first cloud can be matched to any point from the second cloud. In practice, each point from the first cloud is matched to the closest point from the second cloud in terms of its extended Euclidean distance (Eq. 1). $P+ computes the Euclidean distance of a candidate point C_i and a template point T_j along with their difference in turning angles $\left(C_{i.\theta} - T_{j.\theta}\right)$:

$$||C_i - T_j|| = \sqrt{(C_{i.x} - T_{j.x})^2 + (C_{i.y} - T_{j.y})^2 + (C_{i.\theta} - T_{j.\theta})^2} \quad (1)$$

The points from the second cloud that have not yet been matched during this process are matched afterward with the closest point from the first cloud, and their extended Euclidean distances are summed up to the overall dissimilarity score. The cloud matching is applied in both directions: from the candidate cloud to the template cloud, and from the template cloud to the candidate cloud. The minimal score of both directions is considered as the similarity score of the two gestures.

4 THE μV GESTURE RECOGNIZER

The μV recognizer extends the $P+ algorithm by computing the Euclidean distance on shapes, *i.e.*, the equivalent to the local shape distance computed by the !FTL algorithm on resampled gesture points. For this purpose, resampled points are converted into a series of "shapes". The dissimilarity between two gestures (two sets of shapes) corresponds to the sum of the Euclidean distances that separate each pair of corresponding shapes.

Three distinct points on a plane determine six distinct ordered triangles depending on their order of consideration. Two ordered triangles are considered similar despite affine transformations

if they have the same shape in common: if a triangle is rotated, scaled or translated, then it still shares the same shape with its original version. The shape of an ordered triangle is defined as the ratio, using ordered complex numbers $a, b, c \in \mathbb{C}$ for its vertices [11]:

$$Shape\,(a, b, c) = \frac{a - c}{a - b} \in \mathbb{C} \tag{2}$$

Since the concept of shape preserves invariance in terms of rotation, translation, and scaling, it matches exactly the requirements for implementing an RST-invariant recognizer; provided that resampled gesture points are converted into a series of shapes. In a vectorial space, the shape of any triangle is defined as the geometric ratio of two ordered vectors that connect its vertices. A triangle, with ordered vertices A, B, C, has six different shapes, such as:

$$Shape\left(\overrightarrow{AB}, \overrightarrow{BC}\right) = \overrightarrow{AB}/\overrightarrow{BC} \tag{3}$$

The geometric ratio $\overrightarrow{u}/\overrightarrow{v}$ between two plane vectors $\overrightarrow{u} = (u_x, u_y)$ and $\overrightarrow{v} = (v_x, v_y) \in \mathbb{R}^2$ has the following two components:

$$Shape\,(\vec{u}, \vec{v}) = \left(\frac{\vec{u} \cdot \vec{v}}{\vec{v} \cdot \vec{v}}, \frac{\vec{u} \times \vec{v}}{\vec{v} \cdot \vec{v}}\right) \in \mathbb{R}^2 \tag{4}$$

where $\cdot$ and $\times$ denote the scalar product and the vectorial product, resp.: $\overrightarrow{u} \cdot \overrightarrow{v} = u_x * v_x + u_y * v_y$, $\overrightarrow{u} \times \overrightarrow{v} = u_x * v_y - u_y * v_x$

The local shape distance is defined as the Euclidean distance in the Cartesian plane $\mathbb{R}^2$ between two such shapes (Eq. 5). In the μV recognizer, the sum of the between-shapes Euclidean distances accounts for the dissimilarity score between two gestures (represented as two series of shapes). This between-shapes measure substitutes the \$P+'s between-points Euclidean distance to satisfy insensitivity in rotation, scaling, and translation since two shapes are identical despite those affine transformations [15].

$$\left|Shape(\vec{a}, \vec{b}) - Shape(\vec{u}, \vec{v})\right|_{\mathbb{R}^2} = \sqrt{\frac{|\vec{a}|^2|\vec{v}|^2 + |\vec{b}|^2|\vec{u}|^2 - 2\left[(\vec{a} \cdot \vec{b})(\vec{u} \cdot \vec{v}) - (\vec{a} \cdot \vec{v})(\vec{b} \cdot \vec{u}) + (\vec{a} \cdot \vec{u})(\vec{b} \cdot \vec{v})\right]}{|\vec{b}|^2|\vec{v}|^2}} \tag{5}$$

4.1 Implementation

The μV algorithm builds upon the \$P+ recognizer with four key modifications described hereafter. First, since the local shape distance is RST-invariant, rotation, scaling and translation of the candidate and template gestures become unnecessary: pre-processing is reduced to resampling only. A gesture G that comprises S strokes is resampled to P equidistantly-spaced points:

$$G = \left\{p_i = (x_i, y_i) \in \mathbb{R}^2 \mid i \in \{0 \ldots P - 1\}\right\} \tag{6}$$

Notice that the \$P+ classifier resamples the candidate and template gestures to N equidistantly-spaced points. Because the μV recognizer employs a between-shapes measure instead of a between-points approach, we focus on creating N shapes, not on manipulating N resampled gesture points: if N shapes are expected by the recognizer, the resampling rate P of a gesture that comprises S strokes is: $P = N + S$.

The resampled gesture points are converted into a series of N shapes. A shape is determined by connecting three consecutive points. To avoid any inconsistency due to the multistroke aspect of some gestures, our μV recognizer connects two consecutive points to the gesture's centroid. The centroid is intrinsically the most articulation invariant point as it lies "at the center" of all points

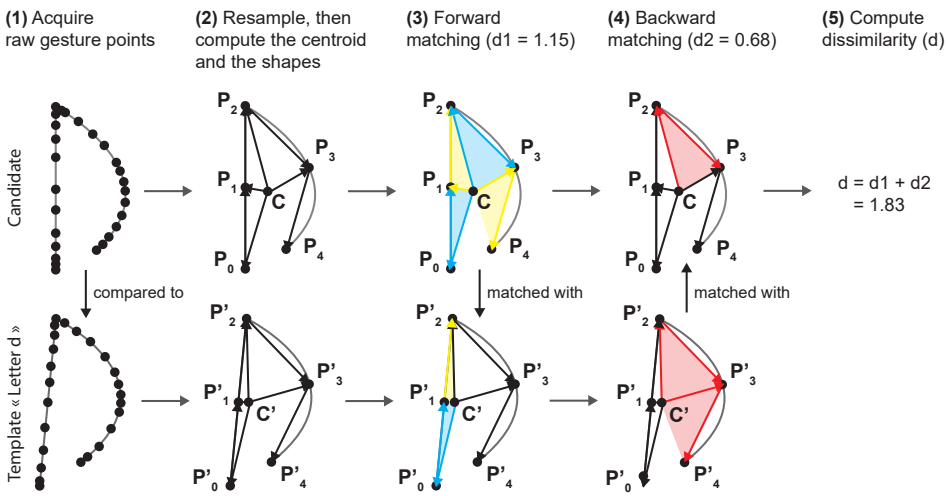

Fig. 2. A candidate is compared to a template letter *d*. The raw gesture points (1) are resampled to five equidistantly-spaced points, their centroid is computed, and the shapes are built between the resampled points and this centroid (2). The shapes are matched to each others in two rounds: each candidate shape is matched to its closest template shape (3), the template shapes remaining after the first round are matched to their closest candidate shape (4). The matched shapes are in the same colour. This procedure (3 and 4) is run twice; during the second run, the order is reversed and the candidate is considered as the second cloud. The class that minimizes the dissimilarity score (d=d_1+d_2) in any cloud matching is then returned.

regardless of the order, number, and direction of the gesture strokes. Considering the centroid c and two resampled points p_i and p_{i+1} that belong to the same stroke, the algorithm calculates the vectors $\vec{u} = \overrightarrow{cp_i}$ and $\vec{v} = \overrightarrow{p_i p_{i+1}}$, then computes the shape of these vectors as shown in Eq. 4. Since a shape connects the centroid of the gesture with two points that belong to the same stroke, in order to create N=8 shapes, μV resamples a two-stroke gesture to P=10 points; whereas it resamples a unistroke gesture to P=9.

Finally, the cloud matching procedure is weighted in two steps: a first *forward matching* where all points from the first cloud are matched to their closest point in the second cloud and are slightly underestimated (w_1=0.95) and a second *backward matching* where all points from the second cloud that have not been matched yet are matched to their closest point in the first cloud and are overestimated (w_2=1.05). The weights w_1 and w_2 have been determined empirically during our evaluation procedure. Similarly to the $P+ recognizer, the cloud matching procedure is run twice; in the first run, the candidate cloud is considered as the first cloud and vice-versa in the second run. The lowest dissimilarity score obtained by the cloud matching runs is stored. The objective is to find the template's gesture class that returns the minimal dissimilarity score.

Figure 2 illustrates the preprocessing and the classification of a candidate gesture against a template labeled as the letter d. In this simple example, the μV recognizer is setup to represent gestures with N=4 shapes. First, the algorithm starts with the raw gesture points for both gestures. Second, since they are one-stroke gestures (S=1), the gestures are resampled to P=N+S=5 points. The centroids of these resampled gestures are computed and their shapes are built: each shape connects two consecutive points that belong to the same stroke to the gesture's centroid. Third, the forward matching matches each shape from the candidate with their closest shape in the template. The first and third shapes from the candidate, coloured in blue, are matched with the first shape from the template, also coloured in blue. The second and fourth shapes from the candidate, coloured

in yellow, are matched with the second shape from the template, also coloured in yellow. This first forward matching provides a dissimilarity score of $d1$=1.15; which represents the underestimated sum of the Euclidean distances between the matched shapes. Fourth, the backward matching matches the third and fourth shapes of the template (*i.e.*, those that are still unmatched) with their closest shape in the candidate (*i.e.*, the third shape of the candidate coloured in red). The overestimated dissimilarity score of this backward matching is $d2$=0.68. The dissimilarity score obtained by the first cloud matching run is d=$d1$+$d2$=1.83. Not shown for the sake of conciseness, the cloud matching procedure will also be run in the reverse order: the template shapes will be compared with the candidate shapes, then the unmatched candidate shapes will be compared with the template shapes, and a second dissimilarity score will be obtained. The lowest dissimilarity score will be stored as the global dissimilarity score between the candidate and the template.

5 EVALUATION

We evaluated μV against five gesture recognizers [17]: (1) $P+, the $-family's most accurate multi-stroke recognizer, (2) $Q, the $-family's fastest multi-stroke recognizer for large datasets, (3) Penny Pincher (PP) and (4) Jackknife (JK), two unistroke recognizers that also use a vector-based representation of gestures, and (5) !FTL, a unistroke recognizer that employs the local shape distance as a dissimilarity measure. We evaluated our recognizer in the traditional user-dependent and user-independent scenarios with gestures from three datasets: one existing and two novel ones.

5.1 Experiment

The $-family recognizers were evaluated with two distinct datasets [2, 29, 37, 39]. The $1 dataset [44] contains 4,800 samples of 16 uni-stroke gestures performed by 10 participants at 3 different speeds (*i.e.*, slow, medium, and fast) with a pen on a pocket PC. The Mixed Multi-stroke Gestures dataset (MMG) [2] contains 3,200 samples of 16 uni-stroke and multi-stroke gestures performed by 20 subjects at the same speed level with a stylus and a finger on a tablet PC. As the pocket and tablet PCs used in these two datasets are less frequent, we conducted an experiment to acquire MMG+, a new MMG dataset with finger gestures performed on a smartphone and a tablet. We share this dataset for reproducibility [6], see Appendix B.

5.1.1 Participants. We recruited 24 participants (6 women, 18 men) between 20 and 29 years (M=21, SD=2.1) from a bachelor's course in human-computer interaction followed by various disciplines: 10 in computer science, 5 in engineering, 5 in linguistics, 2 in communication and 1 in psychology. Twenty participants were right-handed and four were left-handed. All participants declared using a daily computer and a smartphone, except one. A quarter (6) of the participants never used a tablet, whereas half the participants (12) used one occasionally, and the last quarter (6) used one multiple times a week.

5.1.2 Apparatus. The dataset was collected both on a smartphone (Huawei P20 Lite with a 5.84 inch touchscreen) and a tablet (Samsung Galaxy Tab A6 with a 10.1 inch touchscreen). The devices ran a web application specifically developed for acquiring gestures from a 2D canvas inspired by [10]. The UI of the application was as clean and simple as possible: the touchscreen could be used to draw gestures with one or multiple fingers, and two buttons were available at the bottom of the screen either to save the gesture or to clear the canvas. Finally, the instructions were provided at the top of the screen, *e.g.*, "Draw the letter P". Both devices ran the web application with the latest version available for the Google Chrome browser.

5.1.3 Task. The experiment was carried out in two separate sessions of 15 to 25 minutes. During each session, participants were asked to produce 5 samples of each gesture class on a given device.

Half subjects started with the smartphone, the other half with the tablet. They were invited to sit in front of the device placed flat on a desk. The course of the experiment was explained, and only that gestures were going to be used to train and test a gesture recognizer was mentioned. They had to fulfill a survey in which we asked several sociodemographic questions about gender, age, handedness, and discipline. Then, the web application was described to them: an instruction was displayed at the top of the screen, a canvas was available for drawing touch gestures with one or multiple fingers, and two buttons for saving the sample or clearing the canvas. The instructions displayed at the top of the screen were randomly chosen by the web application such that each participant would produce five samples of each class per session.

5.1.4 Rotated Mixed Multi-stroke Gesture set. The experiment was originally not designed to evaluate the performance of a rotation-invariant recognizer. The GPSR method [27] consists of generating synthetic gestures from real data for training a gesture recognizer. Inspired by this method and synthetic gestures [9, 10], we created a second dataset filled with synthetically rotated gestures from the gestures sampled during our experiment. All P points of a gesture $G = \left\{p_i = (x_i, y_i) \in \mathbb{R}^2\right\}$ were randomly rotated by an angle α between $-360°$ and $+360°$ to create a synthetically rotated gesture: $G' = \left\{p'_i = (x'_i, y'_i) \in \mathbb{R}^2\right\}$

$$x'_i = \cos(\alpha) * x_i - \sin(\alpha) * y_i, y'_i = \sin(\alpha) * x_i + \cos(\alpha) * y_i \tag{7}$$

5.1.5 Datasets. The final datasets consist of 16 uni-stroke and multi-strokes gestures from the MMG gesture set. Each gesture class was performed 5 times by 24 participants on each device. In total, each dataset includes 24 (participants) × 16 (gesture classes) × 5 (samples per gesture class) × 2 (devices) = 3,840 gesture samples. The MMG+ dataset consists of the raw sampled gestures, and the RMMG+ consists of synthetically rotated gestures.

5.2 Design

Stroke gesture recognizers are evaluated in one or two distinct scenarios traditionally found in the literature [1, 22, 29, 30, 33, 36, 37, 39, 44]: the *user-dependent* scenario includes only gestures performed by the same user for both training and testing, whereas the *user-independent* scenario takes into account gestures performed by different users for training and testing. User independence is fundamental when a recognizer is trained by one participant, *e.g.*, the UI prototyper, then used by multiple (independent) end users.

5.2.1 User-Dependent. In this condition, the recognizers were trained and tested with gestures performed by the same user. Our evaluation algorithm randomly chose $T = [1, 2, 4]$ samples per gesture class for training the recognizers, then it chose one candidate per gesture class among the remaining samples for testing. We ran the evaluation with a maximum of T=4 templates per gesture class since the evaluation tool required at least one candidate for testing, and each gesture class was performed 5 times per participant in our MMG+ dataset. The procedure was repeated R=100 times for each value of T, and the results were averaged for each participant and each recognizer. In total, we performed 2 (datasets) × 2 (devices) × 24 (participants) × 3 (values for T) × 100 (repetitions) × 6 (recognizers) = 172,800 recognition trials with the MMG+ and the RMMG+ dataset, and 20 (participants) × 4 (values for T) × 100 (repetitions) × 6 (recognizers) = 48,000 recognition trials with the MMG dataset.

5.2.2 User-Independent. In this condition, the recognizers were trained with gestures performed by a set of users, then tested with gestures performed by an independent user. Our evaluation algorithm randomly chose $P = [1, 2, 4, 8]$ participants and $T = [1, 2, 4]$ samples per gesture class for each of these participants to train the recognizers, then it chose one candidate per gesture class

Effect	F	p	η_p^2
Speed × Rates	$F_{2,57} = 0.70$	.5	.02 (–)
Modality × Rates	$F_{1,18} = 1.17$	.29	.06 (–)
Device × Rates	$F_{2,65} = 2.96$	.06	.08 (–)

Table 2. Effect of speed, modality, and device on the recognition rate of μV with their effect size: (L)arge, (M)edium, or (S)mall, and (–) when there is no significant effect size.

produced by the independent participant for testing. The values for P and T are inspired by the \$Q algorithm [39]. The procedure was repeated R=100 times for each value of T and P, and the results were averaged for each independent participant and each recognizer. In total, we performed 2 (devices) × 2 (datasets) × 24 (participants) × 3 (values for T) × 4 (values for P) × 100 (repetitions) × 6 (recognizers) = 691, 200 recognition trials with the MMG+ and the RMMG+ datasets, and 20 (participants) × 4 (values for T) × 4 (values for P) × 100 (repetitions) × 6 (recognizers) = 192, 000 recognition trials with the MMG dataset.

5.2.3 Apparatus. The evaluation was performed on a desktop computer equipped with an Intel Core i5-6600K CPU and 16GB RAM with the web browser Google Chrome. We used publicly-available Javascript versions of the \$P+[1], \$Q[2], !FTL[3], and Jackknife[4] recognizers, and translated the Penny Pincher algorithm from Swift[5] to Javascript. The reader can refer to appendix A for open access to the recognizers and the datasets. The Jackknife (JK) recognizer was evaluated with the inner product as a dissimilarity measure since it performed the most accurate results on the \$1 dataset [30]. The !FTL recognizer was evaluated with the normalized local shape distance (NLSD) as a dissimilarity measure since the NLSD was significantly faster than the traditional local shape distance but not significantly different in accuracy [33]. All recognizers were set up to resample gestures to N=8 points. We expect all recognizers to achieve high accuracy since as few as N=6 points are sufficient to attain a high recognition rate for Euclidean and angular recognizers [34]. We performed the evaluation in user-dependent and user-independent scenarios with touch samples from the MMG, the MMG+, and the RMMG+ datasets. We expected to observe similar results for the μV algorithm, and a large decrease in accuracy for all other recognizers on the RMMG+ dataset.

5.3 Results and Discussion

5.3.1 Effect of Speed, Modality, and Device. As finger and pen gestures exhibit differences for the features analyzed in [31, 32], we studied the effect of the modality (*i.e.*, pen and finger) on the recognition accuracy of our μV recognizer. We also studied the effect of the device (*i.e.*, tablet PC, smartphone, and tablet), and the effect of speed (*i.e.*, slow, medium, and fast) on its recognition rate. The values obtained by the one-way ANOVAs computed for evaluating these effects are summarized in table 2. The three properties had no significant effect on the recognition rate; the algorithm performed similarly with slow, medium, and fast gestures, whether they were sampled on a tablet PC, a a smartphone, or a tablet, using a finger or a pen. For concision, and because these properties had little to no effect on μV's recognition accuracy, we report and analyze only the results obtained with samples from the MMG+ and the RMMG+ datasets acquired with a smartphone. The reader can refer to appendix C for the entire tables of results.

[1]See http://depts.washington.edu/acelab/proj/dollar/pdollarplus.js
[2]See http://depts.washington.edu/acelab/proj/dollar/qdollar.js
[3]See https://github.com/jperezmedina/FTL/
[4]See https://github.com/ISUE/Jackknife
[5]See https://fe9lix.github.io/PennyPincher/

Effect	F	p	η^2
Templates × Rates	$F_{2,69} = 23.64$	<.001	.40 (L)
Templates × Time	$F_{2,69} = 235.03$	<.001	.87 (L)
Participants × Rates	$F_{3,92} = 39.91$	<.001	.56 (L)
Participants × Time	$F_{3,92} = 988.43$	<.001	.97 (L)

Table 3. Effect of the number of templates and participants on the recognition rate and the average recognition time of μV with their effect size: (L)arge, (M)edium, or (S)mall, and (–) when there is no significant effect size.

Effect	F	p	η^2
Recognizers × Rates (UD)	$F_{5,138} = 58.69$	<.001	.68 (L)
Recognizers × Time (UD)	$F_{5,138} = 44826.79$	<.001	.99 (L)
Recognizers × Rates (UI)	$F_{5,138} = 26.61$	<.001	.49 (L)
Recognizers × Time (UI)	$F_{5,138} = 3045.63$	<.001	.99 (L)

Table 4. Effect of the recognizers on the recognition rates and the average recognition time of the gesture samples from the MMG+ dataset acquired with a smartphone. The effect sizes are labeled: (L)arge, (M)edium, or (S)mall, and (–) when there is no significant effect size. (UD) stands for user-dependent and (UI) for user-independent scenarios.

5.3.2 Effect of the Number of Templates and Participants. Template-matchers compare each training template with a candidate gesture in order to classify this candidate. The more templates are available, the slower but the more accurate the recognizers are expected to be. We studied the effect of the number of templates (T=[1, 2, 4]) and the number of participants (P=[1, 2, 4, 8]) on the recognition accuracy and the recognition time of the μV algorithm. Table 3 summarizes the values obtained by the one-way ANOVAs computed for evaluating these effects. As expected, there was a significant effect of the number of templates and the number of participants on the recognition rate with a large effect size, and a significant effect on the average recognition time with a larger effect. Tukey's HSD (and Games-Howell) post-hoc analyses revealed that the μV algorithm was significantly more accurate (and slower) when more templates or participants were provided for training. In the following, we used Tukey's HSD post hoc analysis when Levene's test [12] did not indicate unequal variances, and Games-Howell otherwise.

5.3.3 Effect of the Recognizers on the MMG+ dataset. Table 4 illustrates the values obtained by the one-way ANOVAs used for evaluating the effect of the recognizers on recognition accuracy and average recognition time with gesture samples from the MMG+ dataset acquired with a smartphone. In both scenarios, the recognizers had a significant effect on the recognition rate with a large effect size and a larger effect size on average recognition time. The effect size on recognition rate is larger in the user-dependent scenario than in the user-independent scenario: we show later than user-independent recognition is always less accurate than user-dependent recognition, even with lots of training (P=8). Table 5 reports the recognition rate and average recognition time achieved by the recognizers in the user-dependent and user-independent scenarios on the MMG+ dataset acquired with a smartphone. The significantly more accurate and significantly faster algorithms are highlighted in green. In both scenarios, post-hoc analysis revealed that the μV recognizer was significantly more accurate than the !FTL algorithm and not significantly inferior to the others. Despite statistical evidence, the $P+ recognizer was always slightly superior to its competitors (up to 6% with T=1 and 2% with T=4) in the user-dependent scenario, and the $Q recognizer was slightly superior (up to 2% with P=1, T = 4 and P=8, T=4) in the user-independent scenario. The PP and Jackknife algorithms were inferior to the μV, $P+ and $Q recognizers with few samples (P=1, T=4) but became significantly equivalent in accuracy when the number of samples increased

Recognizers	Rates [%], M (SD)		Time [μs], M (SD)	
	T=1	T=4	T=1	T=4
μV	86 (5)	95 (3)	1.76 (0.13)	3.70 (0.20)
$P+	92 (4)	97 (1)	2.04 (0.15)	3.88 (0.25)
$Q	87 (5)	96 (2)	34.70 (0.55)	38.65 (0.55)
PP	88 (6)	96 (3)	0.35 (0.05)	0.73 (0.94)
JK	85 (6)	96 (2)	2.25 (0.08)	4.90 (0.24)
!FTL	64 (8)	82 (6)	1.24 (0.07)	3.73 (0.42)
Recognizers	**Rates [%], M (SD)**		**Time [μs], M (SD)**	
	P=1, T=4	P=8, T=4	P=1, T=4	P=8, T=4
μV	62 (5)	89 (6)	1.98 (0.13)	27.26 (1.56)
$P+	61 (3)	91 (4)	2.29 (0.18)	32.05 (1.96)
$Q	64 (4)	91 (5)	35.89 (0.57)	89.51 (1.40)
PP	43 (4)	84 (9)	0.37 (0.05)	6.56 (0.06)
JK	46 (5)	84 (10)	1.74 (0.08)	28.08 (1.93)
!FTL	36 (3)	85 (10)	1.37 (0.08)	85.79 (6.60)

Table 5. Results for the user-dependent and user-independent scenarios with gestures performed on a smartphone from the MMG+ dataset.

Recognizers	Rates [%], M (SD)		Time [μs], M (SD)	
	T=1	T=4	T=1	T=4
μV	86 (5)	95 (3)	1.76 (0.13)	3.70 (0.20)
$P+	92 (4)	97 (1)	2.04 (0.15)	3.88 (0.25)
$Q	87 (5)	96 (2)	34.70 (0.55)	38.65 (0.55)
PP	88 (6)	96 (3)	0.35 (0.05)	0.73 (0.94)
JK	85 (6)	96 (2)	2.25 (0.08)	4.90 (0.24)
!FTL	64 (8)	82 (6)	1.24 (0.07)	3.73 (0.42)

Table 6. Results for the user-dependent scenario with gestures performed on a smartphone from the MMG+ dataset.

Recognizers	Rates [%], M (SD)		Time [μs], M (SD)	
	P=1, T=4	P=8, T=4	P=1, T=4	P=8, T=4
μV	62 (5)	89 (6)	1.98 (0.13)	27.26 (1.56)
$P+	61 (3)	91 (4)	2.29 (0.18)	32.05 (1.96)
$Q	64 (4)	91 (5)	35.89 (0.57)	89.51 (1.40)
PP	43 (4)	84 (9)	0.37 (0.05)	6.56 (0.06)
JK	46 (5)	84 (10)	1.74 (0.08)	28.08 (1.93)
!FTL	36 (3)	85 (10)	1.37 (0.08)	85.79 (6.60)

Table 7. Results for the user-independent condition with gestures performed on a smartphone from the MMG+ dataset.

Effect	F	p	η^2
Recognizers × Rates (UD)	$F_{5,138} = 339.31$	<.001	.92 (L)
Recognizers × Time (UD)	$F_{5,138} = 11071.23$	<.001	.99 (L)
Recognizers × Rates (UI)	$F_{5,138} = 95.284$	<.001	.77 (L)
Recognizers × Time (UI)	$F_{5,138} = 2655.12$	<.001	.99 (L)

Table 8. Effect of the recognizers on the recognition rates and the average recognition time of the gesture samples from the RMMG+ dataset acquired with a smartphone. The effect sizes are labeled: (L)arge, (M)edium, or (S)mall, and (–) when there is no significant effect size. (UD) stands for user-dependent and (UI) for user-independent.

Recognizers	**Rates [%], M (SD)**		**Time [μs], M (SD)**	
	T=1	T=4	T=1	T=4
μV	87 (5)	95 (3)	1.80 (0.13)	3.77 (0.26)
$P+	29 (2)	57 (5)	2.50 (0.15)	6.15 (0.26)
$Q	19 (2)	39 (4)	36.60 (1.11)	41.91 (1.62)
PP	26 (2)	55 (4)	0.35 (0.05)	0.57 (0.07)
JK	23 (3)	55 (6)	1.84 (0.01)	7.26 (0.47)
!FTL	64 (8)	82 (6)	1.27 (0.01)	3.85 (0.36)
Recognizers	**Rates [%], M (SD)**		**Time [μs], M (SD)**	
	P=1, T=1	P=8, T=4	P=1, T=1	P=8, T=4
μV	62 (5)	89 (6)	2.00 (0.15)	28.90 (1.97)
$P+	23 (1)	73 (5)	2.56 (0.18)	41.27 (1.25)
$Q	17 (1)	60 (4)	38.03 (1.20)	93.37 (1.14)
PP	15 (1)	59 (5)	0.34 (0.04)	6.63 (0.19)
JK	13 (1)	60 (5)	1.96 (0.08)	35.54 (1.15)
!FTL	36 (3)	70 (7)	1.34 (0.01)	85.19 (7.33)

Table 9. Results for the user-dependent and user-independent scenarios with gestures performed on a smartphone from the RMMG+ dataset.

(P=8, T=4). In the user-independent scenario, post hoc analysis revealed that the μV recognizer was significantly faster than $Q and Jackknife, significantly slower than PP and not significantly inferior to $P and !FTL. In the user-independent scenario, post hoc analysis revealed that the μV recognizer was significantly slower than PP, but significantly faster than all other algorithms.

5.3.4 Effect of the Recognizers on the RMMG+ dataset. Table 8 reports the values obtained by the one-way ANOVAs used for evaluating the effect of the recognizers on recognition rate and average recognition time with gesture samples from the RMMG+ dataset acquired with a smartphone. The recognizers had a significant effect with a large effect size on recognition rate and average recognition time in both scenarios. For the same reason, the effect size on recognition rate is again larger in the user-dependent than in the user-independent scenario.

Table 9 reports the recognition rate and average recognition time achieved by the recognizers in the user-dependent and user-independent configurations on the RMMG+ dataset acquired on a smartphone. Because the gestures were synthetically rotated, only the μV recognizer was expected to produce similar results since it is the sole ARST-invariant algorithm. In both scenarios, post-hoc

Class	Precision [%]	Recall [%]	F1-Score [%]
Fork	82.87	80.91	81.88
D	88.94	90.50	89.71
X	86.15	91.74	88.85
I	94.40	88.88	91.55
Note	77.29	68.62	72.70
!	98.83	88.23	93.23
N	78.86	76.75	77.79
*	84.24	97.81	90.52
H	79.45	82.58	80.99
Line	89.17	98.45	93.58
6-star	79.68	77.00	78.32
Arrow	82.62	78.58	80.55
Null	79.89	94.64	86.64
5-star	76.56	74.59	75.56
P	77.51	76.34	76.92
T	96.54	84.95	90.38
Micro	84.41	84.41	84.41
Macro	84.56	84.41	84.32

Table 10. Per-class precision, recall and F1-scores obtained by the μV recognizer with gestures performed on a smartphone from the MMG+ dataset. Results have been aggregated, then averaged for the user-dependent and user-independent scenarios in all configurations P=[1, 2, 4, 8] and T=[1, 2, 4]. Multi-class micro- and macro-averaged measures are also shown.

analysis revealed the μV recognizer was significantly more accurate than all other algorithms due to its invariance to rotation. In both scenarios, the μV recognizer was significantly slower than Penny Pincher, but significantly faster or equivalent to others.

5.3.5 Multi-Class F1-Scores. In the previous sections, we evaluated the accuracy of our μV recognizer in terms of its recognition rate, *i.e.*, the ratio of correctly predicted candidates to the total number of candidates. The accuracy provides a good measure of a model's performance when false positives and false negatives have the same value. In some particular contexts of use, practitioners might wish differentiating the cost of false positives and false negatives, *i.e.*, some gestures might be critical and should not be confused with another by the recognizer. The F1-score is a weighted measure of precision and recall, often used in machine learning for evaluating the performance of binary classification [18]. It can be extended to multiclass classification [26], *i.e.* gesture recognition, with the use of multiclass micro- and macro-averaged F1-scores.

First, we computed per-class prediction, recall and F1-scores. For each gesture class, per-class prediction is the ratio of correctly predicted candidates to the total number of predicted candidates, and per-class recall is the ratio of correctly predicted candidates to the total number of expected candidates. Table 10 illustrates the scores obtained by each gesture class. The μV recognizer showed great performance (>90%) in classifying the letter I, the exclamation point (!), the asterisk (*), the line and the letter T. However, it showed lower performance (<80%) in classifying the halfnote (note), the letter N, the 6-start, the 5-star, and the letter P.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{8}$$

Recognizers	Micro [%]	Macro [%]
μV	84.41	84.32
$P+	86.12	85.99
$Q	86.09	85.91
PP	76.11	75.82
JK	79.16	80.29
!FTL	60.20	59.95

Table 11. Multi-class micro- and macro-averaged F1-scores obtained by the recognizers with gestures performed on a smartphone from the MMG+ dataset. Results have been aggregated, then averaged for the user-dependent and user-independent scenarios in all configurations.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{9}$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{10}$$

Multi-class F1-scores can be computed in multiple ways [26]. Table 10 reports the micro-averaged and macro-averaged multi-class F1-scores. A macro-averaged F1-score treats all classes equally and calculates their average, whereas a micro-averaged F1-score aggregates the contributions of all classes to compute the average metric; which is similar to measuring the accuracy (recognition rate) of the recognizer.

Table 11 illustrates the micro-averaged and macro-averaged F1-scores for all recognizers when aggregating, then averaging the results obtained in all scenarios (user-dependent and user-independent) and all configurations (P=[1, 2, 4, 8] and T=[1, 2, 4]) with gestures performed on a smartphone from the MMG+ dataset. The μV, $P+ and $Q recognizers show the highest multiclass F1-scores of all recognizers. The same F1-scores have been obtained for all recognizers with gestures performed on a smartphone from the RMMG+ dataset. The μV recognizer shows by far the highest multiclass F1-scores since it is the only recognizer satisfying rotation invariance.

5.3.6 Confusion Matrix. The recognition rate of our μV recognizer does not indicate how well (or not) the algorithm can classify a given gesture class. The confusion matrix is a good way to understand per-class strengths and limitations of a gesture recognizer [38]. Table 12 illustrates the confusion matrix of the μV recognizer with gestures performed on a smartphone from the MMG+ dataset. The results have been aggregated, then averaged for the user-dependent and user-independent scenarios in all configurations (P=[1, 2, 4, 8] and T=[1, 2, 4]). D, X, the asterisk (*), the line, and the null symbol were the most accurately recognized classes (>90%). At the opposite, the halfnote (Note), the 6-star, the 5-star, and the letter P were the least accurately recognized classes (<80%). The 5-star and 6-star symbols were the most confused classes: 11.16% of the time, a 6-star was confused with a 5-star, and 11.51% vice-versa. The letter P and the halfnote were sometimes confused (7.17% and 7.35%) since the halfnote is merely a 180° rotation of the letter P, a sign that our μV recognizer can differentiate slight changes in stroke production. Notice how the exclamation point (!) was sometimes confused with the line (9.74%) but the line was almost never confused with an exclamation point (0.74%).

In sum, the evaluation of our μV recognizer highlights interesting findings: (1) when rotation invariance is not required, the μV algorithm is significantly more accurate than !FTL, and does not significantly differ from the others; (2) in terms of recognition time, μV is significantly faster than its predecessors, except Penny Pincher in all conditions, and Jackknife and !FTL when few

%	Fork	D	X	I	Note	!	N	*	H	Line	6-star	Arrow	Null	5-star	P	T
Fork	80,92	0,01	7,27	0,00	0,80	0,00	0,90	3,53	0,07	0,00	0,01	0,00	5,94	0,58	0,00	0,01
D	0,33	90,50	0,00	0,08	0,38	0,00	0,26	0,00	0,89	0,00	1,43	1,09	2,08	1,60	1,38	0,01
X	1,89	0,00	91,75	0,00	0,09	0,00	0,16	5,91	0,07	0,00	0,00	0,01	0,02	0,04	0,04	0,06
I	0,12	0,13	0,20	88,88	0,56	0,21	1,64	0,02	6,05	1,35	0,02	0,44	0,01	0,04	0,29	0,08
Note	3,98	0,42	2,64	0,14	68,62	0,01	4,00	1,08	1,16	0,00	0,61	6,29	1,11	2,37	7,35	0,27
!	0,05	0,00	0,06	0,61	0,58	88,22	0,03	0,00	0,02	9,74	0,00	0,60	0,00	0,00	0,07	0,03
N	1,56	0,15	0,17	0,67	1,55	0,00	76,75	2,26	9,98	0,00	2,05	0,29	1,48	0,75	2,34	0,02
*	0,09	0,00	1,78	0,00	0,00	0,00	0,24	97,81	0,00	0,00	0,00	0,00	0,01	0,08	0,00	0,00
H	0,43	1,85	0,62	3,02	0,33	0,03	7,55	0,69	82,58	0,00	0,57	0,51	0,04	0,45	1,07	0,26
Line	0,00	0,00	0,00	0,56	0,00	0,74	0,00	0,00	0,00	98,46	0,00	0,26	0,00	0,00	0,00	0,00
6-star	0,30	0,77	0,28	0,00	0,36	0,00	0,85	0,15	0,30	0,00	76,99	0,23	7,10	11,16	1,15	0,38
Arrow	0,19	2,33	0,53	0,07	6,41	0,06	1,61	0,60	1,07	0,85	0,27	78,59	1,04	1,03	3,79	1,61
Null	0,33	0,40	0,02	0,00	0,03	0,00	0,07	0,16	0,00	0,00	2,48	0,20	94,65	1,14	0,56	0,00
5-star	2,86	0,65	0,07	0,00	0,97	0,00	0,29	2,57	0,13	0,00	11,51	0,83	4,17	74,60	1,40	0,00
P	0,20	4,25	0,11	0,12	7,17	0,00	2,43	1,06	0,59	0,00	0,67	2,32	0,84	3,59	76,34	0,34
T	4,43	0,32	1,04	0,03	0,95	0,01	0,58	0,30	1,07	0,02	0,03	3,48	0,03	0,05	2,75	84,96

Table 12. Confusion matrix of the μV recognizer on the MMG+ dataset with gestures performed on a smartphone. Results have been aggregated, then averaged for the user-dependent and user-independent scenarios in all configurations P=[1, 2, 4, 8] and T=[1, 2, 4]. The matrix can be read as follows: the letter D (row) was recognized 2.08% of the time as the null symbol (column).

templates per gesture class are loaded. This suggests that μV scales well when the number of comparisons increases: the more templates, the more comparisons are required by them.

6 CONCLUSION AND FUTURE WORK

We motivated and implemented the μV algorithm, an ARST-invariant 2D multistroke gesture recognizer, by combining two techniques: the $P+'s flexible cloud matching to satisfy articulation-invariance and the !FTL's local shape distance to support RST-invariance. μV is significantly more accurate than its predecessors when rotation invariance is necessary and does not significantly differ from them when it is not. μV is also significantly faster than its predecessors, except Penny Pincher. As it satisfies ARST-invariant recognition, we hope that the μV algorithm could foster the use of multi-stroke gestures in multiple contexts of use, thus facilitating the incorporation of new modalities and new devices in gesture-based interaction. For this purpose, we share the pseudo-code of our μV recognizer in Appendix B.

We plan to pursue our research project in two areas. We would like to extend and release Gester, our software specifically developed to compare recognizers in consistent circumstances, as well as an in-depth analysis of existing 2D stroke gesture recognizers. Based on our MMG+ dataset, we would like to perform the first device-independent evaluation: when the recognizers are trained with gestures performed on a first device (*e.g.*, a smartphone), then tested with gestures performed on a second (independent) device (*e.g.*, a tablet), and vice versa. User- and device-independence are fundamental properties for a stroke recognizer as a UI prototyper is likely to train a recognizer with gestures performed on his own device and multiple (independent) end-users to interact with this recognizer by the intermediate of their own (independent) devices. Moreover, we would like to study the effect of particular gesture classes and participants on recognition rates to better understand the behavior of existing techniques.

We publicly share two novel datasets with finger gestures based on the MMG gesture set, respectively called MMG+, a dataset with raw gestures, and RMMG+, a dataset with synthetically rotated gestures.

ACKNOWLEDGMENTS

The work is supported by the UCLouvain Fonds Speciaux de Recherche under Grant No.: 61273304 and by FRIA under convention No.: FC 31773.

REFERENCES

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

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

[3] Caroline Appert, Emmanuel Pietriga, Éléonore Bartenlian, and Rafael Morales González. 2018. Custom-Made Tangible Interfaces with Touchtokens. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces* (Castiglione della Pescaia, Grosseto, Italy) *(AVI '18)*. Association for Computing Machinery, New York, NY, USA, Article Article 15, 9 pages. https://doi.org/10.1145/3206505.3206509

[4] Adrien Coyette, Sascha Schimke, Jean Vanderdonckt, and Claus Vielhauer. 2007. Trainable Sketch Recognizer for Graphical User Interface Design. In *Proc. of 11th IFIP TC 13 International Conference on Human-Computer Interaction, INTERACT '07* (Rio de Janeiro, Brazil) *(Lecture Notes in Computer Science)*, Maria Cecília Calani Baranauskas, Philippe A. Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa (Eds.), Vol. 4662. Springer, 124–135. https://doi.org/10.1007/978-3-540-74796-3_14

[5] James Herold and Thomas F. Stahovich. 2012. The One Cent Recognizer: A Fast, Accurate, and Easy-to-Implement Handwritten Gesture Recognition Technique. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, Karan Singh and Levent Burak Kara (Eds.). The Eurographics Association. https://doi.org/10.2312/SBM/SBM12/039-046

[6] Kasper Hornbæk, Søren S. Sander, Javier Andrés Bargas-Avila, and Jakob Grue Simonsen. 2014. Is Once Enough? On the Extent and Content of Replications in Human-Computer Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 3523–3532. https://doi.org/10.1145/2556288.2557004

[7] 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* (Funchal, Madeira, Portugal) *(IUI '04)*. Association for Computing Machinery, New York, NY, USA, 156–160. https://doi.org/10.1145/964442.964472

[8] 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* (Atlanta, Georgia, USA) *(CHI '97)*. ACM, New York, NY, USA, 35–42. https://doi.org/10.1145/258549.258574

[9] 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

[10] Luis A. Leiva, Daniel Martín-Albo, and Radu-Daniel Vatavu. 2018. GATO: Predicting Human Performance with Multistroke and Multitouch Gesture Input. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Barcelona, Spain) *(MobileHCI '18)*. Association for Computing Machinery, New York, NY, USA, Article 32, 11 pages. https://doi.org/10.1145/3229434.3229478

[11] James A. Lester. 1996. Triangles I: Shapes. *Aequationes mathematicae* 52 (1996), 30–54. https://doi.org/10.1007/BF01818325

[12] Howard Levene. 1960. Robust tests for equality of variances. In *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, Ingram Olkin and Harold Hotelling et al. (Eds.). Stanford University Press, Palo Alto, CA, USA, 278–292.

[13] Yang Li. 2010. Protractor: A Fast and Accurate Gesture Recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. ACM, New York, NY, USA, 2169–2172. https://doi.org/10.1145/1753326.1753654

[14] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. 2009. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive Mob. Comput.* 5, 6 (2009), 657–675. https://doi.org/10.1016/j.pmcj.2009.07.007

[15] Lorenzo Luzzi and Paolo Roselli. 2020. The shape of planar smooth gestures and the convergence of a gesture recognizer. *Aequationes mathematicae* 94 (2020), 219–233. https://doi.org/10.1007/s00010-020-00712-7

[16] Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2021. Two-Dimensional Stroke Gesture Recognition: A Survey. *ACM Comput. Surv.* 54, 7, Article 155 (July 2021), 36 pages. https://doi.org/10.1145/3465400

[17] Mehdi Ousmer, Arthur Sluÿters, Nathan Magrofuoco, Paolo Roselli, and Jean Vanderdonckt. 2020. Recognizing 3D Trajectories as 2D Multi-stroke Gestures. *Proc. ACM Hum. Comput. Interact.* 4, ISS (2020), 198:1–198:21. https://doi.org/10.1145/3427326

[18] David M.W. Powers. 2011. valuation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies* 2, 1 (2011), 37–63. https://arxiv.org/abs/2010.16061

[19] 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.

[20] James Reaver, Thomas F. Stahovich, and James 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* (Vancouver, British Columbia, Canada) *(SBIM '11)*. ACM, New York, NY, USA, 103–108. https://doi.org/10.1145/2021164.2021183

[21] James Rush Rhyne and Catherine 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

[22] 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

[23] Ugo Braga Sangiorgi, François Beuvens, and Jean Vanderdonckt. 2012. User Interface Design by Collaborative Sketching. In *Proceedings of the Designing Interactive Systems Conference* (Newcastle Upon Tyne, United Kingdom) *(DIS '12)*. Association for Computing Machinery, New York, NY, USA, 378–387. https://doi.org/10.1145/2317956.2318013

[24] Alex Shaw, Jaime Ruiz, and Lisa Anthony. 2021. A Survey on Applying Automated Recognition of Touchscreen Stroke Gestures to Children's Input. *Interacting with Computers* 32, 5-6 (2021), 524–547. https://doi.org/10.1093/iwc/iwab009

[25] Michael Sherman, Gradeigh Clark, Yulong Yang, Shridatt Sugrim, Arttu Modig, Janne Lindqvist, Antti Oulasvirta, and Teemu Roos. 2014. User-Generated Free-Form Gestures for Authentication: Security and Memorability. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services* (Bretton Woods, New Hampshire, USA) *(MobiSys '14)*. Association for Computing Machinery, New York, NY, USA, 176–189. https://doi.org/10.1145/2594368.2594375

[26] Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management* 45, 4 (2009), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

[27] Eugene M. Taranta, Mehran Maghoumi, Corey R. Pittman, and Joseph J. LaViola. 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* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 873–885. https://doi.org/10.1145/2984511.2984525

[28] Eugene M. Taranta, Corey R. Pittman, Jack P. Oakley, Mykola Maslych, Mehran Maghoumi, and Joseph J. LaViola. 2020. Moving Toward an Ecologically Valid Data Collection Protocol for 2D Gestures In Video Games. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3313831.3376417

[29] 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* (Halifax, Nova Scotia, Canada) *(GI '15)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 195–202. http://dl.acm.org/citation.cfm?id=2788890.2788925

[30] 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* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 5850–5861. https://doi.org/10.1145/3025453.3026002

[31] Huawei Tu, Xiangshi Ren, and Shumin Zhai. 2012. A Comparative Evaluation of Finger and Pen Stroke Gestures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 1287–1296. https://doi.org/10.1145/2207676.2208584

[32] Huawei Tu, Xiangshi Ren, and Shumin Zhai. 2015. Differences and Similarities between Finger and Pen Stroke Gestures on Stationary and Mobile Devices. *ACM Trans. Comput.-Hum. Interact.* 22, 5, Article Article 22 (Aug. 2015), 39 pages. https://doi.org/10.1145/2797138

[33] 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* (Boulder, CO, USA) *(ICMI '18)*. Association for Computing Machinery, New York, NY, USA, 125–134. https://doi.org/10.1145/3242969.3243032

[34] Radu-Daniel Vatavu. 2011. The Effect of Sampling Rate on the Performance of Template-Based Gesture Recognizers. In *Proceedings of the 13th International Conference on Multimodal Interfaces* (Alicante, Spain) *(ICMI '11)*. Association for Computing Machinery, New York, NY, USA, 271–278. https://doi.org/10.1145/2070481.2070531

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

[36] 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* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 4667–4679. https://doi.org/10.1145/3025453.3025941

[37] 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* (Santa Monica, California, USA) *(ICMI '12)*. ACM, New York, NY, USA, 273–280. https://doi.org/10.1145/2388676.2388732

[38] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2014. Gesture Heatmaps: Understanding Gesture Performance with Colorful Visualizations. In *Proceedings of the 16th International Conference on Multimodal Interaction* (Istanbul, Turkey) *(ICMI '14)*. Association for Computing Machinery, New York, NY, USA, 172–179. https://doi.org/10.1145/2663204.2663256

[39] 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* (Barcelona, Spain) *(MobileHCI '18)*. ACM, New York, NY, USA, Article 23, 12 pages. https://doi.org/10.1145/3229434.3229465

[40] 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* (Copenhagen, Denmark) *(MobileHCI '15)*. Association for Computing Machinery, New York, NY, USA, 217–226. https://doi.org/10.1145/2785830.2785885

[41] Hiroaki Sakoe Wenjie Cai, Seiichi Uchida. 2014. Comparative performance analysis of stroke correspondence search methods for stroke-order free online multi-stroke character recognition. *Frontiers of Computer Science volume* 8 (2014), pages773—-784. https://doi.org/10.1007/s11704-014-3207-6

[42] Don Willems, Ralph Niels, Marcel van Gerven, and Louis Vuurpijl. 2009. Iconic and multi-stroke gesture recognition. *Pattern Recognition* 42, 12 (2009), 3303–3312. https://doi.org/10.1016/j.patcog.2009.01.030

[43] 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* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1083–1092. https://doi.org/10.1145/1518701.1518866

[44] 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* (Newport, Rhode Island, USA) *(UIST '07)*. ACM, New York, NY, USA, 159–168. https://doi.org/10.1145/1294211.1294238

[45] 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

[46] Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Anderson, and Xiang Cao. 2012. Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review. *Foundations and Trends® in Human–Computer Interaction* 5, 2 (2012), 97–205. https://doi.org/10.1561/1100000012

[47] 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.

A ONLINE RESOURCES

The μV recognizer, implemented in Javascript, and the two additional datasets, MMG+ and RMMG+ stored in a JSON format, are available on GitHub [6] to foster their incorporation into gesture-based interfaces and related research.

B PSEUDOCODE

We provide the complete pseudocode for the μV recognizer; which extends the $P+ algorithm with the local shape distance of the !FTL recognizer. This metric requires the conversion of resampled points into a series of shapes and reduces the preprocessing to resampling only. We highlight changes and additions to the original $P+ recognizer. The μV algorithm uses two data structures called Point and Gesture that can be used to store 2D points (x, y, id) or 2D shapes $(\overrightarrow{x}, \overrightarrow{y}, id)$ that belong to an indexed stroke. The UVRecognizer constructor receives as a parameter the number of shapes P required to represent each gesture, see section 4.1.

```
Point(int x, int y, int strokeId)
1: x ← x
2: y ← y
3: strokeId ← strokeId
```

```
Gesture(Points shapes, string name)
1: shapes ← shapes
2: name ← name
```

```
UVRecognizer(int P)
1: numShapes ← P
2: templates ← []
```

```
addTemplate(Points points, String name)
1: points ← preprocess(points)
2: template ← Gesture(points, name)
3: Append(templates, template)
```

```
recognize(Points points)
1: bestScore ← +∞
2: bestTemplate ← -1
3: points ← preprocess(points)
4: candidate ← Gesture(points)
5: c ← candidate.shapes
6: for each template in templates do
7:     t ← template.shapes
8:     score ← CloudMatching(c, t, bestScore)
9:     if score < bestScore then
10:        bestScore ← score
11:        bestTemplate ← t
12: return templates[bestTemplate]
```

[6] https://github.com/nmagrofuoco/uv

```
PREPROCESS(POINTS points)
1: return SHAPE(RESAMPLE(points))
```

```
RESAMPLE(POINTS points)
 1: resampledPoints ← [points[0]]
 2: numStrokes ← points[points.length-1].strokeId
 3: numPoints ← this.numShapes + numStrokes
 4: int ← PATHLENGTH(points)/numPoints
 5: d ← 0.0
 6: for i ← 1 to i ← points.length do
 7:     p_{i-1} ← points[i-1]
 8:     p_i ← points[i]
 9:     if p_i.strokeId = p_{i-1}.strokeId then
10:         d2 ← EUCLIDEANDISTANCE(p_{i-1}, p_i)
11:         if (d + d2) ≥ int then
12:             pX ← p_{i-1}.x + ((int − d/d2) ∗ (p_i.x − p_{i-1}.x)
13:             pY ← p_{i-1}.y + ((int − d/d2) ∗ (p_i.y − p_{i-1}.y)
14:             p ← POINT(pX, pY, p_{i-1}.strokeId)
15:             APPEND(resampledPoints, p)
16:             INSERT(points, i, p)
17:             d ← 0.0
18:         else
19:             d ← d + d2
20: while resampledPoints.length ≤ numPoints do
21:     APPEND(resampledPoints, points[points.length-1])
22: return resampledPoints
```

```
SHAPE(POINTS points)
 1: shapes ← []
 2: centroid ← CENTROID(points)
 3: for i ← 0 to i ← points.length-1 do
 4:     p_i ← points[i]
 5:     p_{i+1} ← points[i+1]
 6:     if p_i.strokeId = p_{i+1}.strokeId then
 7:         vA ← POINT(p_i − centroid)
 8:         vB ← POINT(p_{i+1} − p_i)
 9:         den ← SCALARPRODUCT(vB, vB)
10:         vX ← SCALARPRODUCT(vA, vB)/den
11:         vY ← (vA.x ∗ vB.y) − (vB.x ∗ vA.y)/den
12:         v ← POINT(vX, vY, vA.strokeId)
13:         APPEND(shapes, v)
14: while shapes > this.numShapes do
15:     POP(shapes)
16: return shapes
```

centroid(Points points)

```
1: dX, dY ← 0.0
2: for i ← 0 to i ← points.length do
3:     dX ← dX + points[i].x
4:     dY ← dY + points[i].y
5: return Point(dX/points.length, dY/points.length, 0)
```

pathLength(Points points)

```
1: length ← 0.0
2: for i ← 1 to i ← points.length-1 do
3:     p_{i-1} ← points[i-1]
4:     p_i ← points[i]
5:     if p_{i-1}.strokeId = p_i.strokeId then
6:         length ← length + euclideanDistance(p_{i-1}, p_i)
7: return length
```

euclideanDistance(Point a, Point b)

```
1: dX ← b.x − a.x
2: dY ← b.y − a.x
3: return Sqrt(dX ∗ dX + dY ∗ dY)
```

scalarProduct(Point a, Point b)

```
1: return a.x ∗ b.x + a.y ∗ b.y
```

cloudMatching(Points shapesA, Points shapesB, int minSoFar)

```
1: dist1 ← cloudDistance(shapesA, shapesB, minSoFar)
2: dist1 ← cloudDistance(shapesB, shapesA, minSoFar)
3: return Min(dist1, dist2)
```

cloudDistance(Points shapesA, Points shapesB, int minSoFar)

```
1:  score ← 0
2:  matchedShapes ← []
3:  for i ← 0 to i ← this.numShapes do
4:      matchedShapes[i] ← false
5:  for i ← 0 to i ← this.numShapes do
6:      index ← −1
7:      minDist ← +∞
8:      for j ← 0 to j ← this.numShapes do
9:          d ← EuclideanDistance(shapesA[i], shapesB[j])
10:         if d < minDist then
11:             minDist ← d
12:             index ← j
13:     score ← score + minDist ∗ 0.95
14:     matchedShapes[index] ← true
15:     if score ≥ minSoFar then
16:         return score
17: for i ← 0 to i ← this.numShapes do
18:     if not matchedShapes[i] then
19:         min ← ∞
20:         for j ← 0 to j ← this.numShapes do
21:             d ← euclideanDistance(shapesA[j], shapesB[i])
22:             if d < minDist then
23:                 minDist ← d
24:                 index ← j
25:         score ← score + minDist ∗ 1.05
26:         matchedShapes[index] ← true
27:         if score ≥ minSoFar then
28:             return score
29: return score
```

Datasets	Rates [%], M (SD)				Time [μs], M (SD)			
	T=1	T=2	T=4	T=8	T=1	T=2	T=4	T=8
MMG (slow)	84 (4)	90 (4)	94 (3)	96 (2)	1.98 (0.53)	2.42 (0.31)	3.73 (0.42)	5.92 (0.76)
MMG (medium)	84 (4)	90 (3)	93 (3)	96 (2)	1.59 (0.09)	2.17 (0.11)	3.37 (0.18)	5.42 (0.45)
MMG (fast)	83 (6)	90 (3)	93 (2)	95 (2)	1.73 (0.22)	2.26 (0.23)	3.48 (0.35)	5.58 (0.43)
MMG (medium, finger)	85 (3)	91 (2)	94 (2)	96 (2)	1.68 (0.23)	2.20 (0.14)	3.34 (0.19)	5.45 (0.34)
MMG (medium, stylus)	82 (5)	88 (4)	92 (3)	95 (2)	1.99 (0.45)	2.66 (0.24)	4.20 (0.49)	6.88 (0.59)
MMG+ (smartphone)	87 (5)	92 (4)	95 (3)	–	1.66 (0.35)	2.34 (0.21)	3.55 (0.33)	–
MMG+ (tablet)	87 (5)	92 (4)	95 (3)	–	1.57 (0.22)	2.22 (0.15)	3.33 (0.34)	–
RMMG+ (smartphone)	87 (5)	92 (3)	95 (2)	–	1.61 (0.43)	2.20 (0.13)	3.31 (0.28)	–
RMMG+ (tablet)	87 (5)	92 (3)	95 (2)	–	1.56 (0.14)	2.26 (0.20)	3.46 (0.34)	–

Table 13. Complete results of the μV recognizer for the user-dependent condition with the MMG, MMG+, and RMMG+ datasets.

Datasets	Rates [%], M (SD)				Time [μs], M (SD)			
	T=1	T=2	T=4	T=8	T=1	T=2	T=4	T=8
MMG (slow)	70 (3)	76 (3)	79 (4)	82 (4)	2.56 (0.13)	1.79 (0.13)	4.08 (0.19)	6.88 (0.40)
MMG (medium)	69 (3)	75 (4)	78 (4)	81 (4)	1.77 (0.11)	2.55 (0.10)	4.10 (0.19)	6.90 (0.37)
MMG (fast)	69 (4)	74 (4)	78 (4)	80 (5)	1.74 (0.11)	2.59 (0.23)	4.14 (0.27)	7.13 (0.74)
MMG (medium, finger)	71 (3)	76 (3)	80 (4)	82 (3)	1.89 (0.46)	2.59 (0.27)	4.15 (0.26)	6.89 (0.33)
MMG (medium, stylus)	68 (4)	74 (4)	77 (4)	80 (4)	1.81 (0.31)	2.62 (0.21)	4.19 (0.26)	7.14 (0.55)
MMG+ (smartphone)	62 (5)	66 (5)	69 (5)	–	1.90 (0.27)	2.69 (0.12)	4.42 (0.29)	–
MMG+ (tablet)	63 (5)	68 (6)	70 (6)	–	1.77 (0.16)	2.54 (0.08)	4.11 (0.18)	–
RMMG+ (smartphone)	62 (5)	67 (6)	70 (5)	–	1.75 (0.21)	2.65 (1.72)	4.30 (0.29)	–
RMMG+ (tablet)	63 (5)	67 (6)	71 (6)	–	1.78 (0.12)	2.61 (0.20)	4.21 (0.33)	–

Table 14. Results of the μV recognizer for the user-independent condition with P=1 participant on the MMG, MMG+, and RMMG+ datasets.

Datasets	Rates [%], M (SD)				Time [μs], M (SD)			
	T=1	T=2	T=4	T=8	T=1	T=2	T=4	T=8
MMG (slow)	79 (4)	83 (4)	86 (4)	87 (4)	2.52 (0.13)	4.06 (0.29)	6.81 (0.53)	11.69 (0.98)
MMG (medium)	78 (4)	81 (4)	85 (4)	87 (4)	2.49 (0.10)	4.12 (0.37)	6.82 (0.36)	11.77 (0.70)
MMG (fast)	77 (5)	81 (4)	83 (5)	85 (4)	2.52 (0.12)	4.07 (0.17)	6.90 (0.40)	11.95 (0.71)
MMG (medium, finger)	80 (3)	83 (4)	86 (3)	87 (3)	2.56 (0.17)	4.07 (0.18)	6.68 (0.24)	11.46 (0.46)
MMG (medium, stylus)	76 (4)	81 (4)	84 (5)	85 (4)	2.54 (0.06)	4.13 (0.11)	7.03 (0.27)	12.16 (0.58)
MMG+ (smartphone)	73 (6)	77 (6)	79 (6)	–	2.62 (0.15)	4.26 (0.25)	7.11 (0.56)	–
MMG+ (tablet)	74 (7)	78 (7)	80 (7)	–	0.25 (0.09)	0.40 (0.24)	0.71 (0.12)	–
RMMG+ (smartphone)	73 (6)	77 (6)	79 (6)	–	2.63 (0.24)	4.24 (0.32)	7.11 (0.65)	–
RMMG+ (tablet)	74 (7)	78 (7)	81 (7)	–	2.56 (0.14)	4.11 (0.24)	7.15 (0.14)	–

Table 15. Results of the μV recognizer for the user-independent condition with P=2 participants on the MMG, MMG+, and RMMG+ datasets.

C EVALUATION (COMPLETE RESULTS)

Tables 13 to 17 illustrate the results (recognition rates and average recognition time) obtained by the μV recognizer on the MMG, the MMG+, and the RMMG+ in all configurations.

Datasets	Rates [%], M (SD)				Time [μs], M (SD)			
	T=1	T=2	T=4	T=8	T=1	T=2	T=4	T=8
MMG (slow)	85 (4)	87 (4)	89 (4)	90 (3)	4.14 (0.37)	6.74 (0.45)	11.59 (0.98)	20.05 (1.98)
MMG (medium)	84 (4)	86 (4)	88 (3)	90 (3)	4.02 (0.18)	6.76 (0.37)	11.64 (0.73)	20.44 (1.38)
MMG (fast)	83 (4)	85 (4)	87 (4)	88 (4)	4.05 (0.18)	6.85 (0.34)	11.93 (0.77)	21.05 (1.59)
MMG (medium, finger)	85 (3)	88 (3)	90 (3)	91 (3)	3.99 (0.16)	6.71 (0.27)	11.30 (0.41)	19.80 (0.77)
MMG (medium, stylus)	82 (5)	85 (4)	88 (3)	89 (4)	4.10 (0.15)	6.96 (0.32)	12.11 (0.58)	21.35 (0.10)
MMG+ (smartphone)	80 (6)	83 (6)	85 (7)	–	4.26 (0.30)	7.37 (0.67)	12.25 (0.99)	–
MMG+ (tablet)	82 (7)	84 (7)	86 (7)	–	4.07 (0.27)	6.75 (0.55)	11.49 (0.92)	–
RMMG+ (smartphone)	80 (6)	83 (6)	85 (6)	–	4.19 (0.38)	6.96 (0.53)	11.91 (1.03)	–
RMMG+ (tablet)	81 (7)	84 (7)	86 (7)	–	4.18 (0.49)	6.93 (0.81)	11.65 (1.02)	–

Table 16. Results of the μV recognizer for the user-independent condition with P=4 participants on the MMG, MMG+, and RMMG+ datasets.

Datasets	Rates [%], M (SD)				Time [μs], M (SD)			
	T=1	T=2	T=4	T=8	T=1	T=2	T=4	T=8
MMG (slow)	88 (3)	90 (4)	91 (3)	92 (3)	6.84 (0.48)	11.56 (0.95)	19.80 (1.74)	34.93 (3.57)
MMG (medium)	87 (4)	89 (3)	91 (3)	92 (2)	6.75 (0.38)	11.62 (0.67)	20.28 (1.39)	36.17 (2.54)
MMG (fast)	86 (4)	88 (4)	89 (4)	91 (3)	0.68 (0.44)	11.93 (0.80)	20.91 (1.56)	37.34 (3.07)
MMG (medium, finger)	89 (3)	91 (3)	92 (3)	94 (2)	6.48 (0.15)	11.22 (0.44)	19.67 (0.69)	34.75 (1.82)
MMG (medium, stylus)	86 (4)	88 (4)	90 (3)	91 (3)	6.93 (0.31)	12.05 (0.56)	21.23 (1.25)	36.31 (2.31)
MMG+ (smartphone)	85 (6)	87 (6)	89 (6)	–	7.30 (0.70)	11.77 (0.72)	20.81 (1.91)	–
MMG+ (tablet)	87 (7)	89 (6)	90 (6)	–	6.81 (0.61)	11.52 (1.09)	19.80 (1.86)	–
RMMG+ (smartphone)	85 (6)	87 (6)	89 (6)	–	6.90 (0.47)	12.00 (1.34)	20.87 (2.23)	–
RMMG+ (tablet)	87 (6)	89 (6)	90 (6)	–	6.84 (0.48)	11.50 (0.90)	19.94 (1.85)	–

Table 17. Results of the μV recognizer for the user-independent condition with P=8 participants on the MMG, MMG+, and RMMG+ datasets.