GestMan: A Cloud-based Tool for Stroke-Gesture Datasets

Nathan Magrofuoco, Paolo Roselli, Jean Vanderdonckt
Université catholique de Louvain
Louvain-la-Neuve, Belgium
{firstname.lastname}@uclouvain.be

Jorge Luis Pérez-Medina
Universidad de las Américas,
Intelligent & Interactive Systems Lab,
Quito, Ecuador
jorge.perez.medina@udla.edu.ec

Radu-Daniel Vatavu
University Stefan cel Mare of Suceava
MintViz Lab | MANSiD Center
Suceava, Romania
radu.vatavu@usm.ro

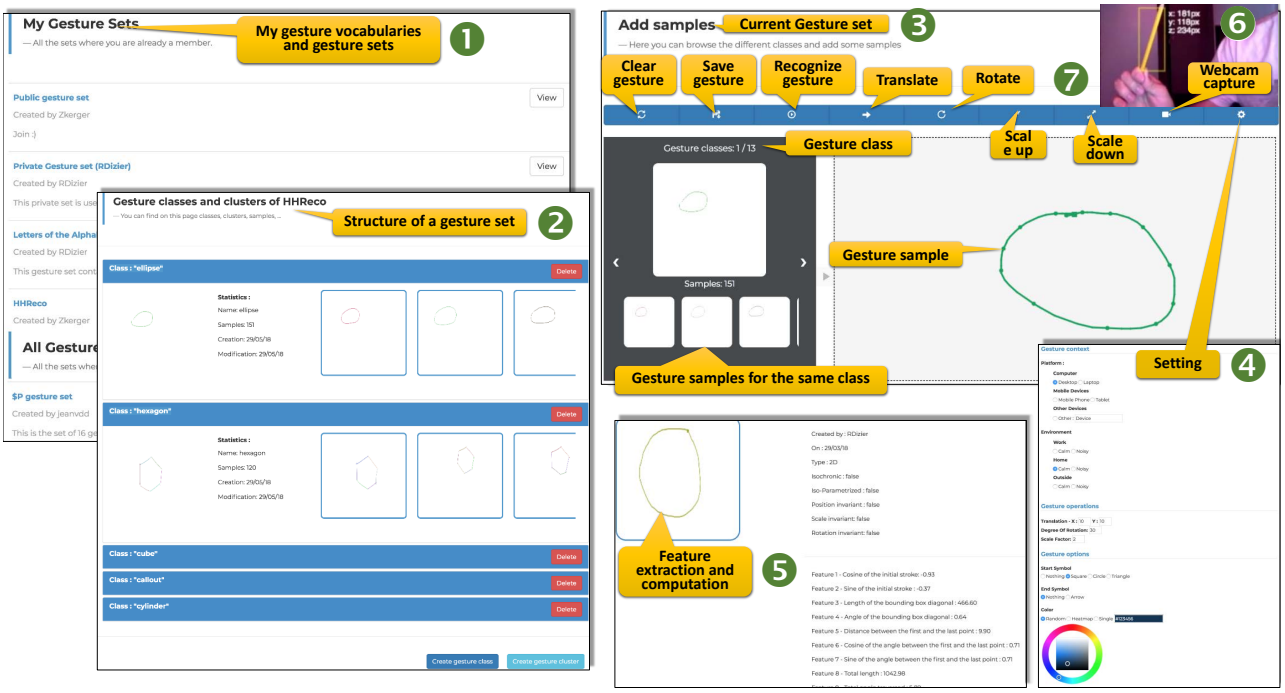

Figure 1: Screenshots of the GestMan application illustrating (1) management of gesture vocabularies and datasets, (2) editing the structure of a gesture set, (3) processing gesture samples, and (4) computation of gesture features.

ABSTRACT

We introduce GestMan, a cloud-based GESTure MANagement tool to support the acquisition, design, and management of stroke-gesture datasets for interactive applications. GestMan stores stroke-gestures at multiple levels of representation, from individual samples to classes, clusters, and vocabularies and enables practitioners to process, analyze, classify, compile, and reconfigure sets of gesture commands according to the specific requirements of their applications, prototypes, and interactive systems. Our online tool enables acquisition of 2-D stroke-gestures via a HTML5-based user interface as well as 3-D touch+air and webcam-based gestures via dedicated mappers. GestMan implements five software quality characteristics of the ISO-25010 standard and employs a new mathematical formalization of stroke-gestures as vectors to support efficient computation of various gesture features.

CCS CONCEPTS

• **Human-centered computing** → **Gestural input**; • **Information systems** → *Data management systems*; • **Theory of computation** → Data structures and algorithms for data management;

KEYWORDS

Stroke-gestures; Gesture sets; Cloud computing; Gesture data management; Tool; Isochronicity; Isometricity; Isoparameterization.

ACM Reference format:
Nathan Magrofuoco, Paolo Roselli, Jean Vanderdonckt, Jorge Luis Pérez-Medina, and Radu-Daniel Vatavu. 2019. GestMan: A Cloud-based Tool for Stroke-Gesture Datasets. In *Proceedings of ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Valencia, Spain, June 18–21, 2019 (EICS '19)*, 6 pages.
https://doi.org/10.1145/3319499.3328227

P. Roselli is also at Università degli Studi di Roma "Tor Vergata", 00173 Roma, Italy.

EICS '19, June 18–21, 2019, Valencia, Spain

ACM ISBN 978-1-4503-6745-5/19/06…$15.00
https://doi.org/10.1145/3319499.3328227

1 INTRODUCTION

The wide availability of touch and motion sensing devices has created an urgent need for advances on good practices for gesture set design towards gesture commands that are efficient to perform [14, 26], easy to recall [2], accurately recognized [15, 21], and a good fit to the functions they execute [9, 28]. Although there are many application scenarios and contexts of use where gesture input is preferable to other interaction modalities, such as to voice input when addressing Internet-of-Things (IoT) devices [12], our present knowledge on designing good gesture sets remains rather scattered in various repositories, which affects its reusability. Practitioners' efforts to systematize gesture sets and data have taken various forms, such as the creation of web pages[1] that collect links to gesture resources available on the web, including public gesture datasets. However, these resources are characterized by *heterogeneity* and are reusable only after hands-on coding efforts: for example, gesture sets that can be downloaded from web sites and repositories come in various file formats, such as `JSON`, `XML`, `CSV`, or `ASCII`, not to mention the various data structures employed to represent them. *Heterogeneity in data representation is present even when it is the same authors that regularly release gesture datasets; see [24–26].* Also, setting up code from repositories into production demands technical skills, time, and effort that sometimes impede adoption by researchers and designers. Consequently, data available in this form do not maximize reusability of available gesture resources [2, 21, 29, 30]. To overcome this limitation, we bring the following practical contributions in the community of Engineering Interactive Computing Systems (EICS):

- GestMan, a cloud-based application for stroke-gesture sets (see Fig. 1 for a screenshot) that implements management of gesture data from (1) acquisition to (2) processing, (3) analysis and classification, (4) composition of gesture sets, and (5) routing of gestures to the next stages of the development process. We also present technical details for the implementation of plugin modules for GestMan regarding the acquisition and storage of various types of gestures, *e.g.*, touch, touch+air, and gestures captured from a video camera.
- A practical discussion of the five ISO-25010 software quality characteristics implemented by GestMan (see Table 1).
- A mathematical formalization of stroke-gestures as vectors implemented by GestMan to compute gesture features.

2 RELATED WORK

The first gesture-based tools were represented by applications primarily aimed at facilitating coding and implementation of gesture recognizers. For example, Gestural Interface Designer (GID) [6], the first gesture design tool to the best of our knowledge, featured dedicated graphical controls as part of its controls toolbox that enabled designers to specify the input modality in the form of pointing and gesture input. The Gesture Design Toolkit (*gdt*) followed as a prototyping tool for designing gesture sets and was updated by Quill [15] to assist designers of pen-based user interfaces in creating and optimizing their stroke-gestures for accurate recognition by the computer. Since one of the design criteria for gesture sets is high recognizability, a variety of stroke-gesture recognition approaches soon followed [17, 19–21, 23, 29]. Among the first ones, the Rubine recognizer [17] required ten to fifteen gesture examples of each *gesture class* to deliver high accuracy. When gesture classes turned out too many, they were structured into *gesture groups*, such as a group for "editing" functions. Gesture classes and groups form a *gesture set*. Once training samples or templates are available, the gesture set can be evaluated for recognition accuracy.

The Gesture and Activity Recognition Toolkit (GART) [16] enabled the development of gesture-based interfaces by providing an abstraction to machine learning algorithms suitable for modeling and recognizing different types of gestures. The toolkit also supported data collection and training of gesture recognizers. Magic [3] was introduced to help designers create motion gestures by providing feedback regarding the internal consistency of the gesture set, the distinguishability of the gesture classes, and highlighting false positives. The follow-up, Magic 2.0 [11], introduced a web service for false positives. UsiGesture [4] is a package for incorporating gestures in graphical user interfaces designed under Eclipse. UsiGesture enables developers to select both the gesture set and the recognizer for a particular application, but also to employ several recognizers at once, yet does not explicitly address the problem of organizing and managing gesture sets. Over the recent years, the level of abstraction of gesture-based user interfaces has been increasing constantly, the Gesture Library [8] and GestIt [18] being two representative examples. For example, GestIt models gestures using temporal and composition operators and connects them to a user interface model in which the feedback is bound to the leafs and nodes of a model decision tree. Several gesture datasets have been made publicly available in the community, including stroke-gestures [25, 26, 29], touch and multi-touch input [24], motion gestures [5], and whole-body gesture datasets [7, 22] as well as textual descriptions of gestures collected from users during elicitation studies, such as the smart rings study of Gheran *et al.* [9].

In conclusion, several environments, tools, and datasets exist that support authoring, design, and development of gesture user interfaces, but the management of gesture sets has been primarily restricted to their own internal usage, *i.e.*, once a gesture set is created in one of these environments, it is directly integrated. 6DMG [5], a dataset of spatio-temporal gestures with position, orientation, acceleration, and angular speed, probably approaches GestMan in that it provides a database for gesture sets, but management operations are not offered via a cloud-based system.

3 GESTURE DATA MANAGEMENT AND PROCESSING WITH GESTMAN

3.1 Software Architecture

We implemented GestMan as a cloud-based `JavaScript` application to enable any online participant to contribute with gesture examples regardless of their operating system, web browser, or input device. The technology used for GestMan was MEAN (`MongoDB`, `Express.js`, `Angular.js`, and `Node.js`) deployed in the form of a `Heroku` application on top of a persistent gesture database; see Fig. 2 for a visual illustration of the architecture. `MongoDB` is structured in terms of *collections* (the equivalent of tables in `SQL` terminology) of `JSON` documents (equivalent to columns in `SQL` terminology) characterized by *schemas*, which can be directly queried based on *fields*

[1] https://sites.google.com/site/adriendelaye/home/pen-and-touch-datasets

Characteristic (factor)	Definition
Interoperability (compatibility)	Degree to which two or more systems, products or components can exchange information and use it
Integrity (security)	Degree to which a system prevents unauthorized access to, or modification of information
Accountability (security)	Degree to which the actions can be traced uniquely to the user
Modifiability (portability)	Degree to which a system can be modified without introducing defects
Adaptability (portability)	Degree to which a system can effectively and efficiently be adapted to evolving hardware, software, and usages

Table 1: ISO/IEC 25010 characteristics and corresponding definitions used to assess software quality for GestMan.

(or SQL rows), ranges, or regular expressions. MongoDB also supports a series of functions to manipulate stored gestures, making them *persistent* and *interoperable* from one application to another. Since MongoDB is a NoSQL document-oriented database that manipulates JSON documents, it relies on schema to structure the gesture data. These schema can be seen as the "blueprints" of the collections from the database, enabling validation of the various fields of the stored gestures. While MongoDB uses a flexible schema that does not enforce any document structure, it prescribes the usage of more rigid schema that are then converted into models; see Fig. 2b. These models are used to instantiate documents in the database, ensuring that each document follows the structure defined in the corresponding Mongoose schema and can be manipulated accordingly.

3.2 Gesture Data Structure

GestMan presents three views for each gesture (see Fig. 1):

(1) An *external view* that depicts each gesture graphically to end-users ③.
(2) A *conceptual view* showing gesture properties that are useful for designers ⑤.
(3) An *internal view* that presents gestures in terms of raw data (see Fig. 3), which can be directly queried from the database, such as *"which gestures are appropriate for navigation with a smartphone?"*

The atomic level of our representation is the *point* with corresponding timestamp and list of properties. All samples can be assigned (*property*, *value*) pairs. This way, an event listener can be implemented on a specific gesture area to receive notification when the associated property changes. A *stroke* represents a list of points and a gesture is represented as a list of strokes.

A *gesture class* ② contains all the gesture samples of the same type. Gestures can be grouped into a *gesture cluster*, which can be decomposed into a series of sub-clusters, *e.g.*, the cluster "Letters" could group all the letters and be further decomposed into lowercase and uppercase letters. Classes and clusters form a *gesture set* ①, *e.g.*, "Letters and Digits," which can be assembled into a *gesture vocabulary*, such as the "Latin alphabet" or "More's alphabet"; see Fig. 4. Gesture sets can be declared as *public* to be accessed by all users without registration required, *application-oriented* when the gesture set can be modified by GestMan registered users only, or *private* when it can be only modified by invited members. Any gesture set can also be marked as *standard*, such as widely employed gesture sets to preserve their consistent usage, such as HHReco

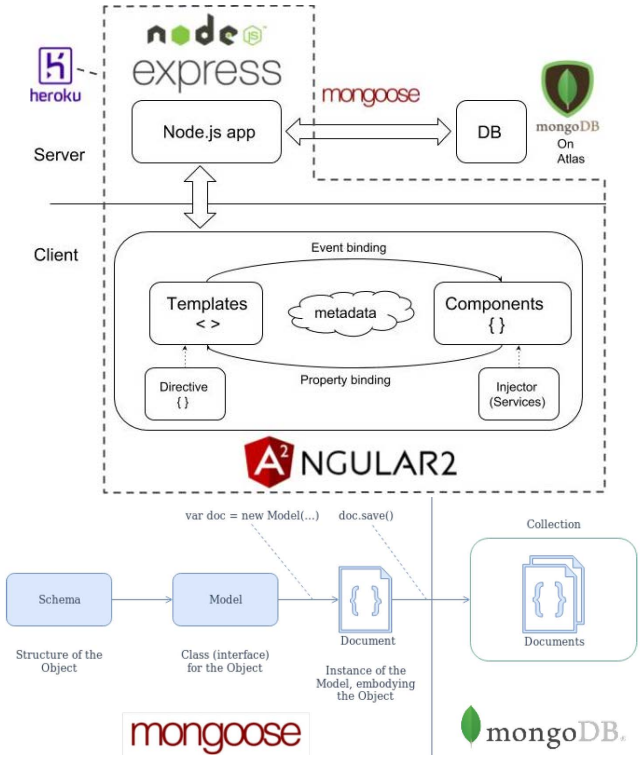

Figure 2: GestMan software architecture.

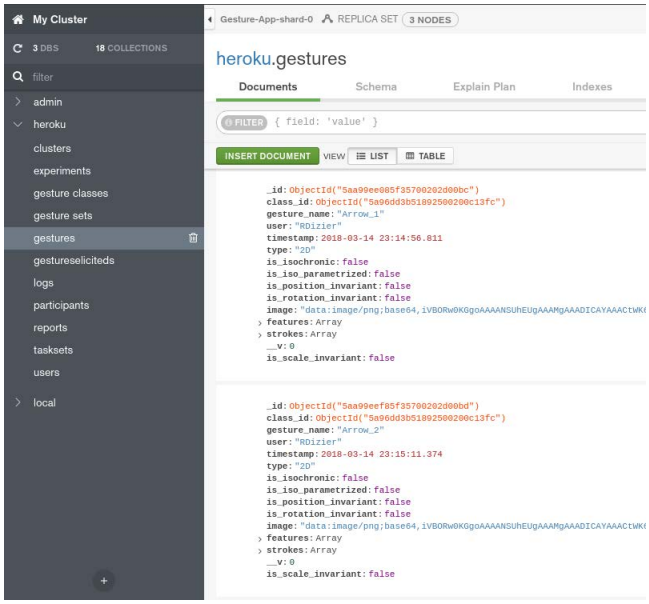

Figure 3: The internal view for gestures stored in GestMan.

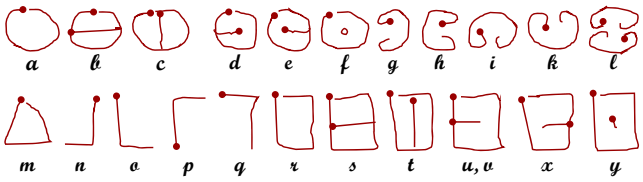

Figure 4: More's alphabet of stroke-gestures.

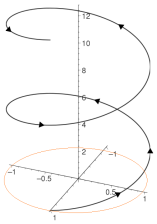

Figure 5: A "helix" 3-D motion gesture.

[10], NicIcon [27], $1 [29], or the difficulty datasets [26], which are unmodifiable to ensure *integrity*. These gesture sets can be duplicated to enable further editing.

3.3 Gesture Acquisition

During acquisition, points are delivered by a sensor and stored as real numbers with timestamps and referred to as Raw Data Points (RDP) [6] or in the form of Euler angles or Tait-Bryan angles. Gestures can be converted into a vector-based representation, as employed by PennyPincher [19] and !FTL [20] gesture recognizers, which reduces the size of the representation that is both structure preserving and coordinates-free. Each gesture gesture sample ③ can be acquired on-demand with contextual information ④ regarding the user, the input device, and the environment. Hence, any gesture can be declared as *user-dependent* or *user-independent*. Similarly, gestures can be attached to a particular platform, *e.g.*, a tablet, or environment, *e.g.*, a smart room. Therefore, gestures can be marked as user-, device-, or environment-dependent under GestMan. A platform-dependent gesture is applicable to only one platform. The current version of GestMan supports acquisition of 2-D gestures (via a HTML5-compatible browser) and 3-D gestures (through finger- or object-tracking ⑥ from video and the 3DTouchPad device); see Fig. 7 and 5 .

3.4 Gesture Processing

GestMan provides the following set of primitives ⑦: acquire, clear, save, recognize, translate, rotate, and scale. Making a gesture position, scale, or rotation invariant is optional [30]. For instance, the "person" gesture from the NicIcon dataset [27] should be rotation-dependent for two classes: a person laying down and standing up, although the symbol remains the same. Similarly, a the arrow ">" gesture can be made scale-dependent when associated to fast forward function, for instance. The starting point of a gesture is marked using a specific symbol and color, see the red circle shown in Fig. 4, which is a recommendation from Chen *et al.* [5].

3.5 Gesture Analysis and Recognition

GestMan computes geometric features for gestures ⑤, such as Rubine's features [17]. If needed, analysis can be conducted at sample level, and gestures can be processed to maintain properties, such as *isometricity* (*i.e.*, the same distance between the points on the gesture path), *isochronicity* (gesture points that are equally spaced in time), and *isoparameterization* (the same amount of points) [20].

In addition to concentrating gesture knowledge into a single repository, GestMan supports conducting experiments in an incremental and collaborative way that relate to the three ACM badges[2] concerning *repeatability* (same team, same experimental setup), *reproducibility* (different team, same experimental setup), and *replicability* (different team and different experimental setup). Gesture set owners can ask participants to acquire new gestures or modify existing gestures. For instance, Fig. 6 presents a snapshot from an experiment conducted using GestMan.

The current version of GestMan implements four recognizers: $1 [1], $P [21], !FTL and !NFTL [20], while PennyPincher [19], and $Q [23] will be integrated in the future. GestMan supports *adaptability* by enabling other recognizers, algorithms, and web services (as recommended in [11]) to be integrated, such as Gestures-a-GoGo [13] for producing synthetic gestures or KeyTime [14] for predicting gesture production time. GestMan enables direct import of code supporting therefore *modifiability*. Also, GestMan captures stroke-gestures as points that can be converted to a vector representation. To this end, we defined vector-based formulae for Rubine's features [17], *e.g.*, the f_1 feature can be computed as follows:

$$f_1 = \cos\alpha = \frac{x_2 - x_0}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}} = \frac{(\vec{u}_2 - \vec{u}_0) \cdot \vec{e}_1}{|\vec{u}_2 - \vec{u}_0|} \quad (1)$$

3.6 Gesture Composition

To support gesture integration, gestures can be *composed* by appending their strokes, *decomposed* from multi-strokes to unistrokes, and *recomposed* at any level. When a gesture class is considered definitive, it can be declared as "standard" to prevent changes or loss of information. All these operations along with the primitives from the processing stage are recorded into a log file that can be replayed, providing thus a design history. Since all operations are tracked, *accountability* is intrinsically supported.

[2] https://www.acm.org/publications/policies/artifact-review-badging

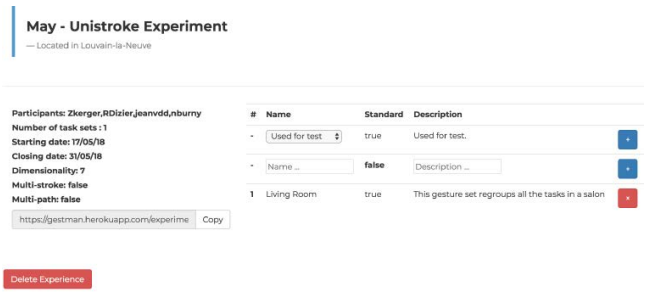

Figure 6: Conducting an experiment on a gesture set.

Figure 7: Touch+air gestures acquired by 3DTouchPad. Source: https://www.microchip.com/DevelopmentTools/ProductDetails/DM160225

3.7 Gesture Routing

To support deployment of gesture vocabularies or sets, GestMan exports `JavaScript` code of a selected gesture recognizer and the gesture set stored in a compatible format. The exported resources can be incorporated in the development life-cycle of the gesture user interface.

4 PLUGIN MODULES FOR GESTMAN

Stroke-gestures are in GestMan as points entered using an `HTML5` canvas or from a video camera using a special module. In the first case, gesture points are directly captured and stored. In the second case, points need to be extracted from the video using image processing techniques. Therefore, a generic software architecture problem arises: how to acquire a wide variety of gesture types using an application running in a web browser? For a stand-alone application, this question is usually solved in a straightforward way by relying on the SDK/API provided by the device vendor. Unfortunately, this solution is tied to a particular software/hardware and not always available in a web browser. In the rest of this section, we exemplify our solution for two gesture types: touch+air gestures and webcam-based 3-D gestures.

4.1 Touch+Air Gestures

Touch+air gestures are 2-D touch multistrokes followed by mid-air gestures performed at a small distance (maximum 30 cm) from the touchpad; see Fig. 7.. To acquire such gestures[3], we developed a 3DTouchPad mapper with two threads (see Fig. 8) [4]:

(1) A `LibWebSockets` client responsible for the polling of the 3DTouchPad device that collects data every 50 ms and creates a `JSON` message sent to a server. `LibWebSockets`[5] is an open-source `C` library for lightweight network protocols: a data structure containing the connection information is created with a protocol setup, an initialisation takes place and the loop for the service is launched; a function callback is executed for each event.

[3]https://www.microchip.com/DevelopmentTools/ProductDetails/DM160225
[4]Code is accessible at https://github.com/gigi199596/3DTouchPad-Mapper.
[5]https://libwebsockets.org/

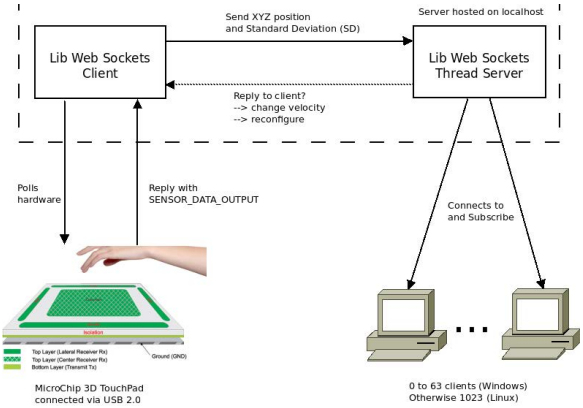

Figure 8: Software architecture of the 3DTouchPad Mapper.

(2) A `LibWebSockets` server that sends messages to connected clients. The server also delivers a local web page to display a log of the messages that were received.

4.2 Webcam-based Gestures

To acquire a 2-D and a half or 3-D gesture, the `tracking.js` `JavaScript API`[6] uses lightweight computer vision techniques to track the color of a pointer, such as an object, a pen, a finger, through the webcam. This pointer color can be configured in the settings box, such as a yellow highlighter (⑥ in Fig. 1).

5 CONCLUSION AND FUTURE WORK

We introduced GestMan, a publicly available[7] cloud-based application for collaboratively managing stroke-gestures, vocabularies, sets, classes, and clusters, which implements five ISO quality characteristics. GestMan as a handy platform for practitioners, researchers, and developers to collaboratively manage their gesture knowledge by fostering reusability, this artifact is evaluated and reusable[8] since it is documented, consistent (GestMan is introduced in this technical note), complete, and permanently exercisable (GestMan can be used by simple login and password). The code is accessible through the Dashboard Heroku (see "Deploy" to clone the project): https://dashboard.heroku.com/apps/gestman.

Although some more sophisticated gestures could be captured, GestMan still does not support other exchange protocols like TUIO[9] or Virtual Reality Peripheral Network (VRPN)[10], which would allow GestMan to support gesture acquisition from a wider variety of devices. We leave such explorations for future work.

GestMan is intended to create a Community of Practice (CoP) around stroke-gesture datasets, where every interested party could contribute by: converting existing datasets and importing them into GestMan, adding new datasets, editing shared datasets, defining standard datasets such as those promoted by software vendors, perform a comparison of stroke-recognizers on some of these datasets.

[6]https://trackingjs.com/
[7]https://gestman.herokuapp.com/
[8]https://www.acm.org/publications/policies/artifact-review-badging
[9]See www.tuio.org
[10]See https://github.com/vrpn/vrpn/wiki

ACKNOWLEDGMENTS

R.-D. Vatavu acknowledges supported from a grant of the Ministry of Research and Innovation, CNCS-UEFISCDI, project no. PN-III-P1-1.1-TE-2016-2173 (TE141/2018), within PNCDI III.

REFERENCES

[1] Lisa Anthony, Radu-Daniel Vatavu, and Jacob O. Wobbrock. 2013. Understanding the Consistency of Users' Pen and Finger Stroke Gesture Articulation. In *Proceedings of Graphics Interface 2013 (GI '13)*. Canadian Information Processing Society, Toronto, Ont., Canada, 87–94. http://dl.acm.org/citation.cfm?id=2532129.2532145

[2] Caroline Appert and Shumin Zhai. 2009. Using strokes as command shortcuts: cognitive benefits and toolkit support. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009, Boston, MA, USA, April 4-9, 2009*. 2289–2298. https://doi.org/10.1145/1518701.1519052

[3] Daniel Ashbrook and Thad Starner. 2010. MAGIC: A Motion Gesture Design Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. 2159–2168.

[4] François Beuvens and Jean Vanderdonckt. 2012. Designing Graphical User Interfaces Integrating Gestures. In *Proceedings of the 30th ACM International Conference on Design of Communication (SIGDOC '12)*. ACM, New York, NY, USA, 313–322. https://doi.org/10.1145/2379057.2379116

[5] Mingyu Chen, Ghassan AlRegib, and Biing-Hwang Juang. 2012. 6DMG: A New 6D Motion Gesture Database. In *Proceedings of the 3rd Multimedia Systems Conference (MMSys '12)*. 83–88.

[6] R. B. Dannenberg and D. Amon. 1989. A Gesture Based User Interface Prototyping System. In *Proceedings of the 2nd Annual ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST '89)*. ACM, New York, NY, USA, 127–132. https://doi.org/10.1145/73660.73676

[7] Simon Fothergill, Helena Mentis, Pushmeet Kohli, and Sebastian Nowozin. 2012. Instructing People for Training Gestural Interactive Systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1737–1746. https://doi.org/10.1145/2207676.2208303

[8] Bruno Galveia, Tiago Cardoso, Vitor Santor, and Yves Rybarczyk. 2015. Towards the creation of a Gesture Library. *EAI Endorsed Transactions on Creative Technologies* 2, 3 (6 2015). https://doi.org/10.4108/ct.2.3.e3

[9] 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)*. ACM, New York, NY, USA, 623–635. https://doi.org/10.1145/3196709.3196741

[10] 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)*. ACM, New York, NY, USA, 156–160. https://doi.org/10.1145/964442.964472 Retrieved September 9, 2017 from https://embedded.eecs.berkeley.edu/research/hhreco/.

[11] D. Kohlsdorf, T. Starner, and D. Ashbrook. 2011. MAGIC 2.0: A web tool for false positive prediction and prevention for gesture recognition systems. In *Face and Gesture 2011*. 1–6. https://doi.org/10.1109/FG.2011.5771412

[12] Myeongcheol Kwak, Youngmong Park, Junyoung Kim, Jinyoung Han, and Taekyoung Kwon. 2018. An Energy-efficient and Lightweight Indoor Localization System for Internet-of-Things (IoT) Environments. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 1, Article 17 (March 2018), 28 pages. https://doi.org/10.1145/3191749

[13] 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

[14] Luis A. Leiva, Daniel Martín-Albo, Réjean Plamondon, and Radu-Daniel Vatavu. 2018. KeyTime: Super-Accurate Prediction of Stroke Gesture Production Times. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 239, 12 pages. https://doi.org/10.1145/3173574.3173813

[15] Allan Christian Long, Jr., James A. Landay, and Lawrence A. Rowe. 1999. Implications for a Gesture Design Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 40–47. https://doi.org/10.1145/302979.302985

[16] Kent Lyons, Helene Brashear, Tracy Westeyn, Jung Soo Kim, and Thad Starner. 2007. GART: The Gesture and Activity Recognition Toolkit. In *Proceedings of the 12th International Conference on Human-computer Interaction: Intelligent Multimodal Interaction Environments (HCI'07)*. Springer-Verlag, Berlin, Heidelberg, 718–727. http://dl.acm.org/citation.cfm?id=1769590.1769671

[17] Dean Rubine. 1991. Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1991, Providence, RI, USA, April 27-30, 1991*, James J. Thomas (Ed.). ACM, 329–337. https://doi.org/10.1145/122718.122753

[18] Lucio Davide Spano, Antonio Cisternino, Fabio Paternò, and Gianni Fenu. 2013. GestIT: A Declarative and Compositional Framework for Multiplatform Gesture Definition. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '13)*. ACM, New York, NY, USA, 187–196. https://doi.org/10.1145/2494603.2480307

[19] Eugene M. Taranta, Andres N. Vargas, and Joseph J. LaViola. 2016. Streamlined and accurate gesture recognition with Penny Pincher. *Computers Graphics* 55 (2016), 130 – 142. https://doi.org/10.1016/j.cag.2015.10.011

[20] 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 2018 on International Conference on Multimodal Interaction (ICMI '18)*. ACM, New York, NY, USA, 125–134. https://doi.org/10.1145/3242969.3243032

[21] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2012. Gestures as point clouds: a $P recognizer for user interface prototypes. In *International Conference on Multimodal Interaction, ICMI '12, Santa Monica, CA, USA, October 22-26, 2012*. 273–280. https://doi.org/10.1145/2388676.2388732

[22] 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)*. ACM, New York, NY, USA, Article 224, 13 pages. https://doi.org/10.1145/3290605.3300454

[23] 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

[24] Radu-Daniel Vatavu, Gabriel Cramariuc, and Doina Maria Schipor. 2015. Touch Interaction for Children Aged 3 to 6 Years: Experimental Findings and Relationship to Motor Skills. *International Journal of Human-Computer Studies* 74 (2015), 54–76. http://dx.doi.org/10.1016/j.ijhcs.2014.10.007

[25] Radu-Daniel Vatavu and Ovidiu-Ciprian Ungurean. 2019. Stroke-Gesture Input for People with Motor Impairments: Empirical Results & Research Roadmap. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 215, 14 pages. https://doi.org/10.1145/3290605.3300445

[26] Radu-Daniel Vatavu, Daniel Vogel, Géry Casiez, and Laurent Grisoni. 2011. Estimating the Perceived Difficulty of Pen Gestures. In *Human-Computer Interaction – INTERACT 2011*, Pedro Campos, Nicholas Graham, Joaquim Jorge, Nuno Nunes, Philippe Palanque, and Marco Winckler (Eds.). Springer, Berlin, 89–106.

[27] Don Willems, Ralph Niels, Marcel van Gerven, and Louis Vuurpijl. 2009. Iconic and multi-stroke gesture recognition. *Pattern Recognition* 42, 12 (2009), 3303 – 3312. New Frontiers in Handwriting Recognition.

[28] 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)*. ACM, New York, NY, USA, 1083–1092. https://doi.org/10.1145/1518701.1518866

[29] 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

[30] Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Andersen, 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