International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 2
ISSN: 2088-5334

Algorithmic Efficiency of Stroke Gesture Recognizers: a Comparative Analysis

Ana Belén Erazo[a1], Jorge Luis Pérez Medina[a2]

[a] *Intelligent and Interactive Systems Lab (SI2-Lab), Universidad de Las Américas (UDLA), Quito, 170504, Ecuador*
E-mail: [1]ana.erazo@udla.edu.ec; [2]jorge.perez.medina@udla.edu.ec

***Abstract*— Gesture interaction is today recognized as a natural, intuitive way to execute commands of an interactive system. For this purpose, several stroke gesture recognizers become more efficient in recognizing end-user gestures from a training set. Although the rate algorithms propose their rates of return there is a deficiency in knowing which is the most recommended algorithm for its use. In the same way, the experiments known by the most successful algorithms have been carried out under different conditions, resulting in non-comparable results. To better understand their respective algorithmic efficiency, this paper compares the recognition rate, the error rate, and the recognition time of five reference stroke gesture recognition algorithms, i.e., $1, $P, $Q, !FTL, and Penny Pincher, on three diverse gesture sets, i.e., *NicIcon, HHReco*, and *Utopiano* Alphabet, in a user-independent scenario. Similar conditions were applied to all algorithms, to be executed under the same characteristics. For the algorithms studied, the method agreed to evaluate the error rate and performance rate, as well as the execution time of each of these algorithms. A software testing environment was developed in JavaScript to perform the comparative analysis. The results of this analysis help recommending a recognizer where it turns out to be the most efficient. *!FTL (NLSD)* is the best recognition rate and the most efficient algorithm for the *HHreco* and *NicIcon* datasets. However, Penny Pincher was the faster algorithm for *HHreco* datasets. Finally, *$1* obtained the best recognition rate for the *Utopiano* Alphabet dataset.**

***Keywords*— gesture interaction; gesture recognition; algorithmic efficiency; stroke analysis.**

I. INTRODUCTION

Gesture-based User interfaces are often recognized for their naturalness and intuitiveness [1], [2], with a wide range of applications, such as diagram design, user interface prototyping, online food ordering, handwriting recognition, in-car interaction, and smart home [3]. Today's operating systems do support programming gesture-based interfaces, even suggesting some design guidelines for appropriate mapping gestures to functions, which is a complex problem [4]. Unfortunately, these guidelines cannot cover all potential usages. In order to collect gestures for other purposes that are not covered, elicitation techniques [5] can be used to inform the design process for new gestures [6]. Subsequently, techniques such as machine learning, template-based pairing, and pattern recognition are used to incorporate these gestures into a gesture recognition process. Gesture recognition algorithms typically try to distinguish the candidate gesture that was drawn, to compare it with reference gestures previously recorded in the system, and to return the closest gesture that has been found, if any.

Nowadays, the main challenge for gesture recognition is to produce a recognition algorithm with the best algorithmic efficiency, e.g., in terms of execution time, recognition rate, and computational complexity. Several such algorithms already exist, but they are all programmed in different languages and tested on different *gesture sets*.

The literature review does not report any analysis in which these algorithms are compared under similar conditions., thus having consequences on hypotheses and/or assumptions about these algorithms. A contribution to addressing this problem, therefore, consists of conducting a comparative analysis of the most prominent gesture recognizers, such as *$1* [7], *$P* [8], *$Q* [9], *!FTL* and *!NFTL* [10] and Penny Pincher [11], under the same experimental conditions, using as assessment parameters their recognition rate and execution time. To comply with this, a case study will be carried out that takes as a reference a set of gestures, which will be used individually, in the evaluation of the algorithms, in order to compare the algorithms, indicated above.

The comparative analysis is performed under the same consistent conditions of execution and with the same reference sets of gestures. The JavaScript programming language is used to guarantee the same execution environment. The study is focusing on three gesture sets: *NicIcon* [12], *HHreco* [13], [14] and *Utopiano* Alphabet [15]. This last dataset was created by us to incorporate other

sets of gestures in our comparative evaluation. As a result of the study, it is expected to obtain a series of statistical tables with the data obtained from the experimentation will be presented and analyzed to identify the most appropriate platform for the recognition of the proposed gesture.

The purpose of this article is to report the results of a comparative study to contribute with the advances in recognition of gestures by being able to make known which are the most efficient algorithms, making the selection process easier when it is necessary to incorporate natural interactions with gestures in software applications. This will help in the creation of new systems that allow a much more sophisticated and simpler human-computer interaction for end users. With correct implementation of a recognition algorithm, a system can be made much more efficient and time and money will be saved in its development. A special approach is towards the implementation of these recognizers in systems that help the prototype in the analysis and requirements gathering phase, since they will show in real-time how the interfaces and views will be displayed in the application being developed, making that these stages of development are much more precise and respond to the user's taste.

The remainder of this paper is structured as follows: the next section reviews some work related to the most known gesture recognition algorithms. The third presents the comparative evaluation under the same conditions of execution. The fourth section concludes the paper by summarizing the findings of the experiment and by discussing some future avenues of this work.

II. Materials and Method

A. Gesture Definition

In general, a gesture is referred to as anybody movement performed to convey some meaning to the audience, such as some thoughts, opinions, ideas, feelings, intentions, or to combine them with speech. In gesture interaction, a gesture is more precisely defined as "any physical movement that a digital system can sense and respond to without the aid of traditional pointing devices, such as a mouse or stylus" [1]. More specifically in the context of 2D gesture recognizers, a *stroke gesture* consists of a sequence of points delineated by a starting point and an ending point [2]. There are two types of gestures [2]: (1) *Uni-stroke* are gestures without any interruption in their line [7], and (2) *Multi-stroke* are gestures with time/space interruption among their strokes [16]. An interruption is detected with the lifting of the input device (e.g., a stylus or a mouse) until touching the surface again for the next stroke of the gesture [8].

A gesture can also have a resampling, which is to transform the figure with n number of points to a specific number of points $p \leq n$. Gestures have several invariance properties such as: translation (when invariant to position), scale (when invariant to size), rotation (when invariant to angle), and articulation (when invariant to the way the stroke have been performed).

B. Current Status of Stroke gesture Recognition Algorithms

In this section, we deliver a brief overview of selected stroke gesture recognizers to provide the study with some background in this field. A pioneer is *GrandMa* [17], which recognizes a candidate gesture from a training set by computing 13 geometric features and comparing them with the reference gestures contained in the training set. Instead of extracting and comparing features, a typical process found in machine learning, Levenshtein-based recognition (LVS) [18] decomposes a gesture into a suite of small directional strokes indicating the eight directions of a compass, and compares gestures with the Levenshtein distance.

$1 [7] showed a quantum leap in the race for the best recognizer by forgetting about complex programming environments, procedures, and recognition processes to simplify the recognition to its maximum. In this way, it opened a new series of recognizers. $1 is a template-based *Uni-Stroke gesture recognizer* that offered a very good recognition rate while keeping the recognition time low. $P [8] extended *$1* to recognize *multi-stroke gestures* by adopting a cloud matching approach instead of a pattern, thus offering flexibility in the way gestures are compared, indifferently from their composition, direction, ordering, and number of strokes. *$Q* [9] is the last member of the *$*-family, which optimizes the recognition time, especially for low-end devices which are computationally less efficient. Until now, these recognizers are point-to-point pattern matcher in that they compare points of the candidate gesture to those of the reference gestures. Penny Pincher [11] follows a point-to-vector pattern matching by transforming points into vectors to be compared. *!FTL* [10] generalizes this approach with a vector-to-vector pattern matching: all gestures are vectorized and compared based on the Lester distance generalized on vectors instead of points. This approach intrinsically satisfy position, scale, and rotation invariances since everything is computed based on vectors. For these reasons, we chose recognizers considered as representative members of these three families.

C. Apparatus

The evaluations were carried out using a MacBook computer with the following characteristics: Processor: 1.1 GHz Intel Core M. RAM memory: 8 GB. SSD: 251 GB. Graphics Card: Intel HD Graphics 5300 1536 MB. Operating system version: macOS 10.14.4 (18E226). Kernel version: Darwin 18.5.0.

D. The Software Experimentation Environment

Figure 1 contains the main screen of the runtime environment of *!FTL* [10]. This environment was developed in JavaScript. The figure shows that the main functionalities of the experimentation environment are: save a gesture, compare a gesture, clean the working area, load gestures, export gestures, compare datasets, export files with the results. In order to save a candidate gesture, the first thing that is done is to draw the gesture within the working area, place a name, choose the number of points for resampling, select the threshold and click on the button to save the gesture. The next step is to compare the gesture, for this another gesture is made with which you are going to undergo the experimentation, and click on the "Compare Gestures" command. This will display the results for each algorithm, showing the name of the resulting gesture, the time used to obtain the result and the distance, as shown in Figure 1.

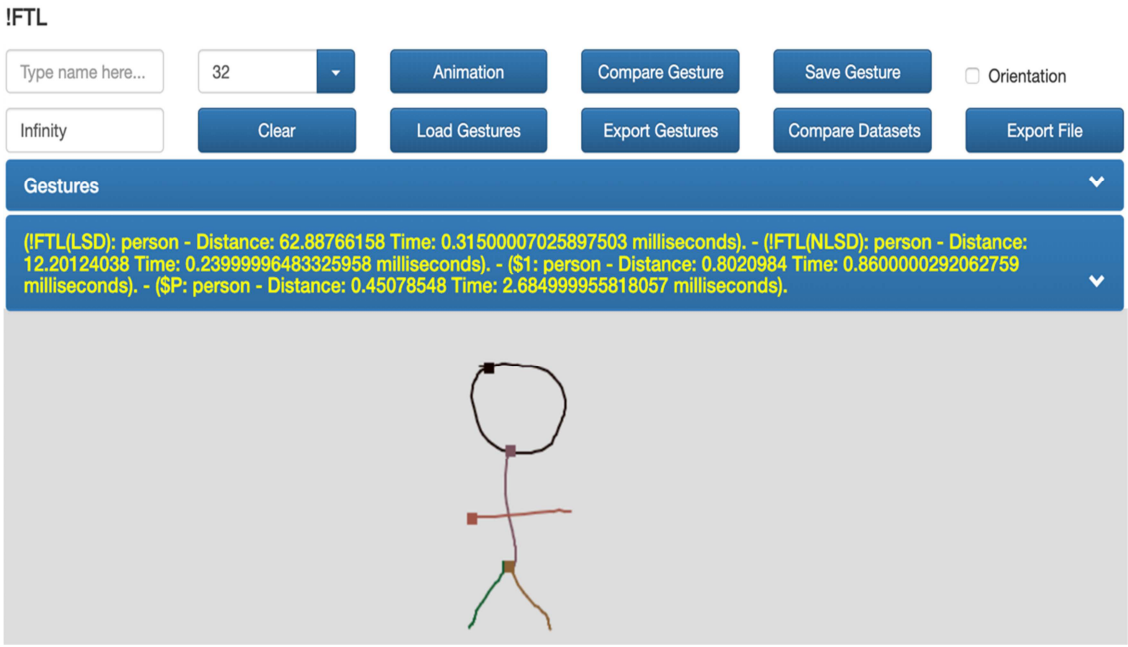

Fig. 1 The interface of the experimentation environment

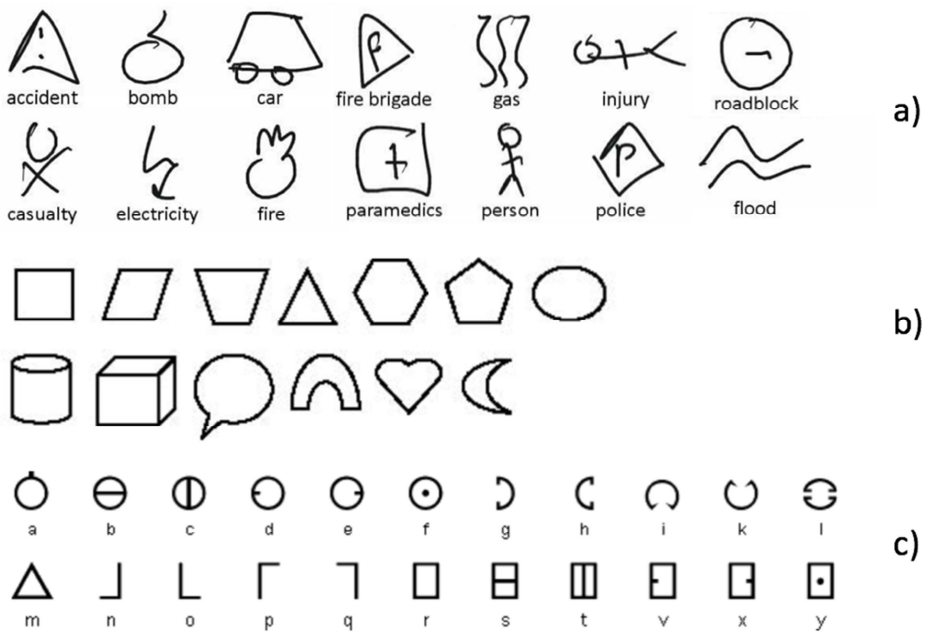

Fig. 2 The datasets used in the experiment: a) *NicIcon* [12]; b) *HHreco* [13, 14] and c) *Utopiano* Alphabet

E. Gesture Recognition Algorithms

As previously mentioned, the algorithms selected were *$1*, *$P*, *$Q*, *!FTL* and Penny Pincher, since they belong to three different families. We are not aware of any software environment that incorporates all these gesture recognizers at once, except in *iGesture* [19], which consists of a toolkit for integrating stroke gesture interaction in user interfaces, but not for comparing algorithms.

F. Datasets

To perform the experiment, a set of datasets and a common execution environment were used. Figure 2 and Table 1 describe the datasets used in the experiment. Note that for each dataset a significant number of gestures is displayed.

TABLE I
CHARACTERISTICS OF THE DATASETS USED IN THE EXPERIMENT

Dataset	**Nro. Repre-sentations**	**Nro. Users**	**Represen-tations by user**	**Total Gestures**
NicIcon [12]	14	32	55	14x32x55=24.640
HHreco[13, 14]	13	19	30	13x19x30 = 7.410
Utopiano Alphabet	22	20	10	22x20x10=4.400

G. Quantitative Measures

1) Recognition Rate: The recognition rate is an indicator that allows to calculating the percentage of effectiveness of an algorithm when it is subjected to an evaluation of gestures of a candidate dataset with respect to the reference gestures previously stored in the system. This means that, in comparison, each candidate gesture will be evaluated with the recognition algorithms against the reference gestures that are previously registered in the system. The result will be indicated showing the closest gesture in distance and the time it took to compare. Then, the number of times in which the algorithm recognized the candidate gestures correctly is added. This sum is divided over the number of gestures compared and with that, the recognition rate is already obtained.

2) Error Rate: The error rate is a factor that allows to calculate the percentage of deficiency of a specific algorithm when it is submitted to the evaluation of the gestures of a candidate dataset with respect to the previously stored reference gestures. This means that, in the comparison, each candidate gesture will be evaluated with the recognition algorithms against the gestures that are in the system and the result will be indicated, showing the closest gesture in distance and the time it took to compare. Then, the number of times the algorithm recognized the candidate gestures incorrectly is added. This sum will be divided by the number of gestures compared, thus enabling to compute the error rate.

3) Recognition Time: The recognition time is an indicator that captures the time in which an algorithm has been worked when it is subjected to an evaluation, in which a candidate dataset is compared with the datasets stored in the system, using the algorithm procedure to indicate the result. The recognition time is used to see the performance of each algorithm evaluated. This factor is an average value since time is measured since the evaluation of a gesture of the candidate dataset begins and ends with the result of that gesture. This time is added for each gesture and finally the average time is computed in milliseconds.

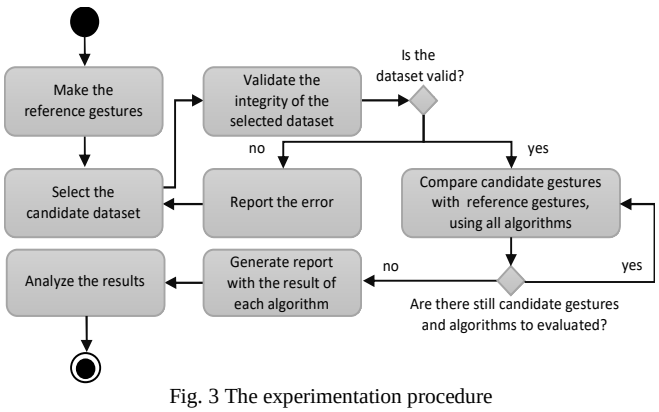

Fig. 3 The experimentation procedure

H. Procedure

Similar conditions were applied to all algorithms, to be executed under the same characteristics. For the algorithms studied, the method agreed to evaluate the error rate and performance rate, as well as the execution time of each of these algorithms. First, a similar execution environment was chosen for the comparison process. For these reasons, the $Q and Penny Pincher algorithms were implemented in the JavaScript language. Next, these algorithms were integrated into the *!FTL* test environment. In parallel, the *Utopiano* dataset was defined. After that, the *Utopiano* dataset was created, with the voluntary participation of 20 people. Once the similar conditions were created for each of the algorithms, the experimentation was carried out with each dataset. For each dataset the experimentation procedure performed is summarized in Figure 3.

III. Results And Discussion

This section presents the results obtained from the comparative evaluation of the *!FTL, $1, $P, $Q* and *PennyPincher* algorithms, performed with the *NicIcon*, *HHreco* and *Utopiano* datasets. For each of the datasets, the recognition rate, the recognition time and the error range of the algorithms were calculated. For each measure, the results will be reported and discussed.

I. Recognition Rate

NicIcon recognition rate analysis: Table 2 and Figure 4 show the average results of the recognition rate for the *NicIcon* dataset. We can appreciate from these results, the most recognized algorithm was *!FTL(LSD)*, followed by !FTL(NLSD). On the other hand, the gestures that had more accurate results were Electricity and Roadblock. It is important to note that some algorithms did not fully recognize certain gestures, such as *!FTL(NLSD)* and $1 with the "Gas" gesture. Similarly, the same situation happens with the algorithm which did not recognize the "Flood" gesture.

TABLE II
NicIcon RECOGNITION RATE (EXPRESSED IN PERCENTAGE VALUES)

	FTL (LSD)	**FTL (NLSD)**	**$1**	**$P**	**$Q**	**Penny Pincher**	**Average**
Gas	0.20	0.00	0.00	0.91	0.51	2.42	0.67
Casualty	1.01	0.00	1.92	3.03	0.30	1.31	1.26
Police	2.93	1.22	0.91	2.83	0.71	0.10	1.45
Fire brigade	6.79	0.30	6.19	2.98	0.71	1.52	3.08
Paramedics	4.34	5.56	8.18	1.82	1.41	11.92	5.54
Accident	6.05	18.63	2.43	5.34	0.85	2.46	5.96
Car	6.05	18.63	2.43	5.34	0.85	2.46	5.96
Person	4.24	0.20	3.03	10.00	18.99	9.70	7.69
Fire	8.91	8.91	19.21	5.23	0.20	8.69	8.53
Injury	9.80	0.10	13.43	11.82	16.57	11.21	10.49
Flood	12.73	7.88	8.28	0.81	37.98	0.00	11.28
Bomb	31.59	35.20	7.10	5.05	0.91	4.35	14.03
Roadblock	14.06	14.67	7.38	26.12	16.42	7.08	14.29
Electricity	28.33	21.22	21.32	15.80	1.52	19.83	18.00
Average	**9.79**	**9.47**	**7.27**	**6.93**	**6.99**	**5.93**	

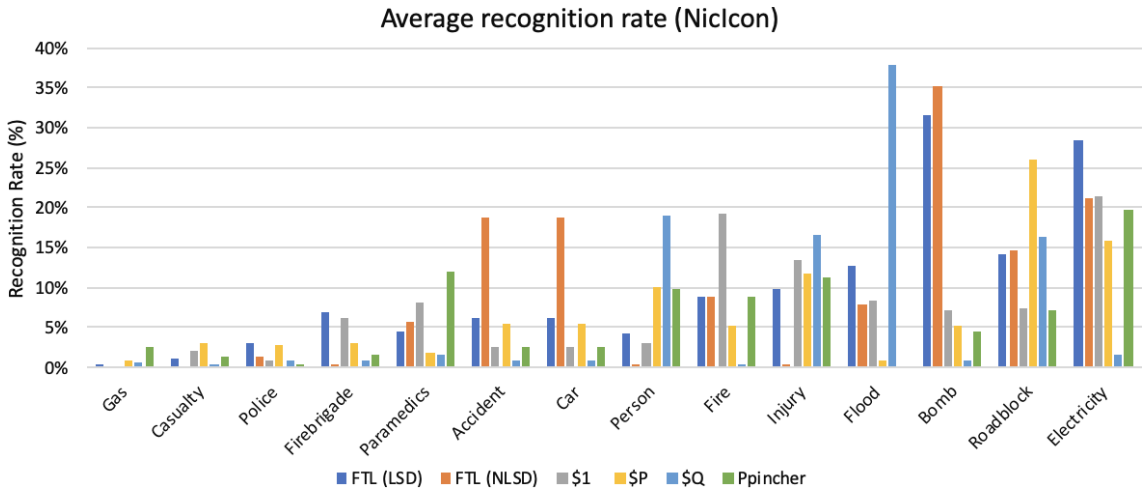

Fig. 4 Recognition rate resulting from the *NicIcon* dataset

HHreco recognition rate analysis: Table 3 and Figure 5 present the results obtained after experimentation with the *HHreco* dataset. The most recognized algorithms was *!FTL(NLSD)*, followed by $1. The most recognized gestures were Ellipse and Arch.

TABLE III
HHRECO RECOGNITION RATE (EXPRESSED IN PERCENTAGE VALUES)

	FTL (LSD)	FTL (NLSD)	$1	$P	$Q	Penny Pincher	Average
Parallelogram	1.96	15,29	4.71	4.51	5.49	3.14	5.85
Hexagon	2.94	18.63	10.39	0.78	1.18	1.37	5.88
Moon	12.35	16.27	4.90	4.12	4.12	4.31	7.68
Pentagon	6.47	14.12	23.14	0.59	1.57	0.39	7.71
Cylinder	0.98	16.47	19.80	6.86	4.12	8.04	9.38
Trapezoid	0.78	12.75	13.14	9.80	11.57	9.41	9.58
Square	13.92	32.16	8.63	4.31	4.31	4.71	11.34
Triangle	12.16	34.90	27.45	7.25	5.88	8.04	15.95
Cube	3.33	15.29	20.00	22.75	11.65	24.12	16.24
Heart	13.14	16.47	34.12	19.41	23.92	16.08	20.52
Callout	28.43	34.31	36.27	12.35	11.76	12.94	22.68
Ellipse	64.12	95.29	15.10	10.00	8.82	4.12	32.91
Arch	6.27	5.10	28.63	50.20	54.51	54.90	33.27
Average	**12.84**	**25.16**	**18.94**	**11.76**	**11.48**	**11.66**	

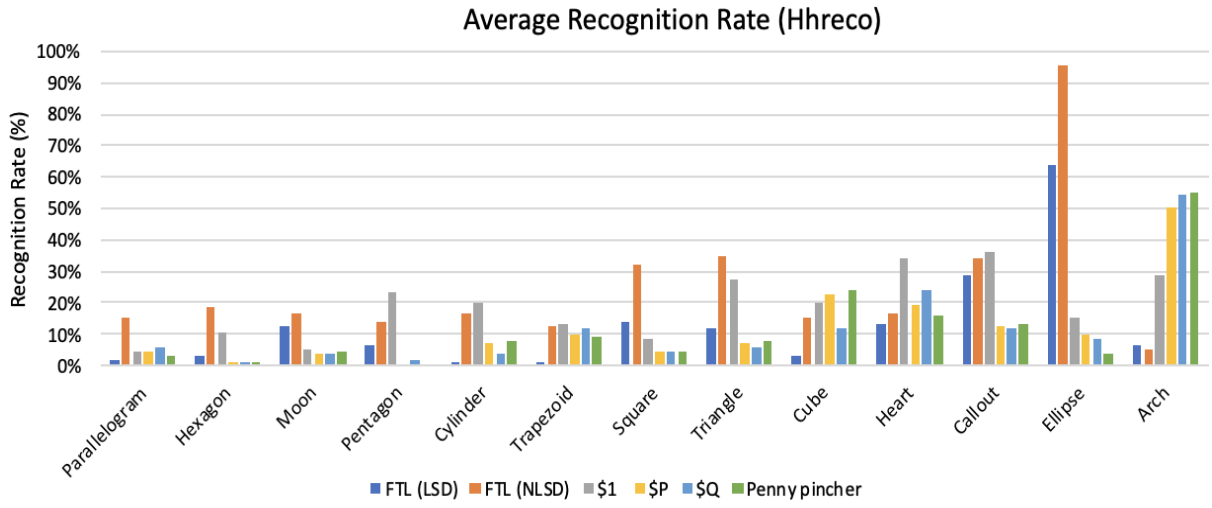

Fig. 5 Recognition rate resulting from the *HHreco* dataset

Utopiano recognition rate analysis: Table 4 and Figure 6 indicate the results obtained after experimentation with the *Utopiano* dataset. You can see that the algorithm that most recognized was *$1* and *!FTL(NLSD*). The most recognized gestures were the letter N and P.

TABLE IV
UTOPIANO RECOGNITION RATE (EXPRESSED IN PERCENTAGE VALUES)

	FTL (LSD)	FTL (NLSD)	$1	$P	$Q	Penny Pincher	Average
B	1.60	3.85	3.90	1.10	0.85	0.95	2.04
E	16.50	23.00	19.00	7.00	7.00	7.00	13.00
S	2.50	6.50	44.00	10.00	13.00	13.00	14.33
Y	2.00	17.33	32.67	12.67	10.67	10.67	14.44
V	5.50	13.50	30.50	14.50	15.00	15.00	15.67
M	5.00	7.00	43.00	17.50	11.00	11.00	16.83
X	7.50	16.50	39.00	13.00	13.50	13.50	17.17
K	23.16	12.11	54.21	7.89	11.05	11.05	19.30
D	30.50	38.00	18.50	15.00	11.50	11.50	21.75
T	4.50	15.00	44.50	23.50	18.50	18.50	21.83
F	43.50	46.50	26.00	9.50	9.50	9.50	24.58
R	9.00	6.50	62.00	25.00	22.00	22.00	24.75
C	16.50	47.00	44.00	14.00	18.00	18.00	25.83
A	23.50	32.00	65.50	13.50	10.50	10.50	26.33
I	24.50	26.50	37.50	25.50	24.00	24.00	27.25
O	33.50	39.50	62.50	14.50	11.50	11.50	29.33
L	31.00	68.00	56.00	12.50	5.50	5.50	30.58
H	20.00	32.50	67.00	20.50	22.00	22.00	31.08
G	17.83	35.17	78.67	20.17	20.83	20.83	32.47
N	20.00	35.00	67.00	28.50	17.00	17.00	32.67
P	33.50	19.50	74.00	35.00	28.50	28.50	37.58
Q	40.50	37.00	70.50	32.00	34.00	34.00	40.75
Average	**18.73**	**26.27**	**47.27**	**16.95**	**15.25**	**17.24**	

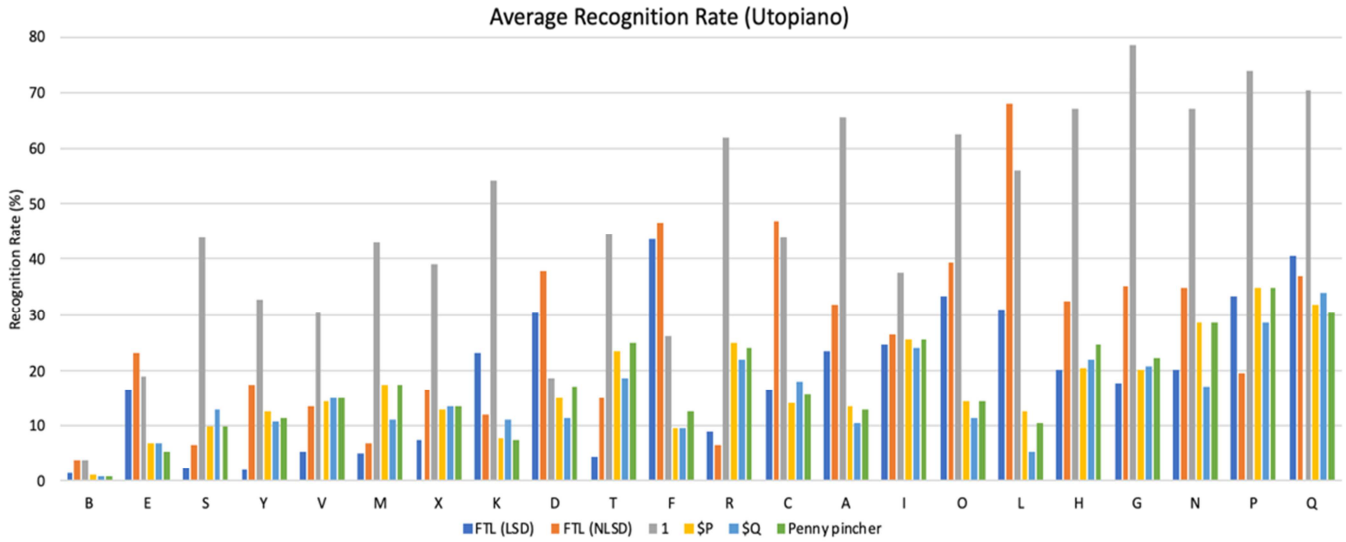

Fig. 6 Recognition rate resulting from the *Utopiano* dataset

J. Recognition Time

NicIcon recognition time analysis: Figure 7 shows the average recognition time results of the *NicIcon* dataset. Technically it is observed that the Penny Pincher algorithms, followed by *!FTL(NLSD)*, are the fastest to recognize *NicIcon* gestures. Gas and Police were the gestures with the best recognition times, that is, the lowest. Despite the speed of these algorithms, it should be borne in mind that some of the cases, these algorithms do not fully recognize gestures, as is the case of Penny Pincher with the Flood and Police gesture. While in the case of *!FTL(NLSD)*, the unrecognized gestures are Gas and Casualty.

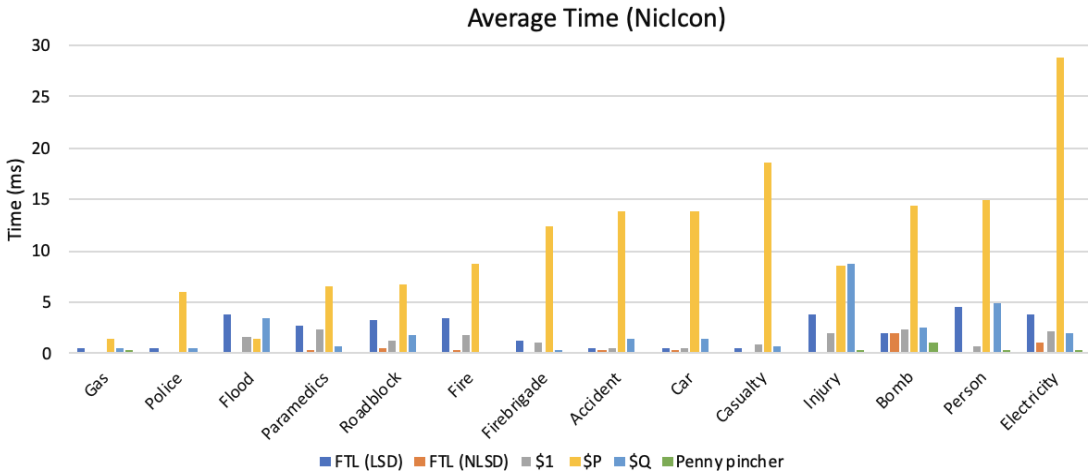

Fig. 7 Average recognition time resulting from the *NicIcon* dataset

HHreco recognition time analysis: Figure 8 presents the results of the average recognition time of the *HHreco* dataset. It is indicated that the Penny Pincher algorithms, followed by !FTL(LSD), are the fastest to recognize *HHreco* gestures. The fastest representations to be recognized were Pentagon and Hexagon. Despite the result obtained by these algorithms, their recognition rate was not very high, therefore, these algorithms cannot be considered very efficient. On the contrary, this shows that they had the least ability to recognize gestures.

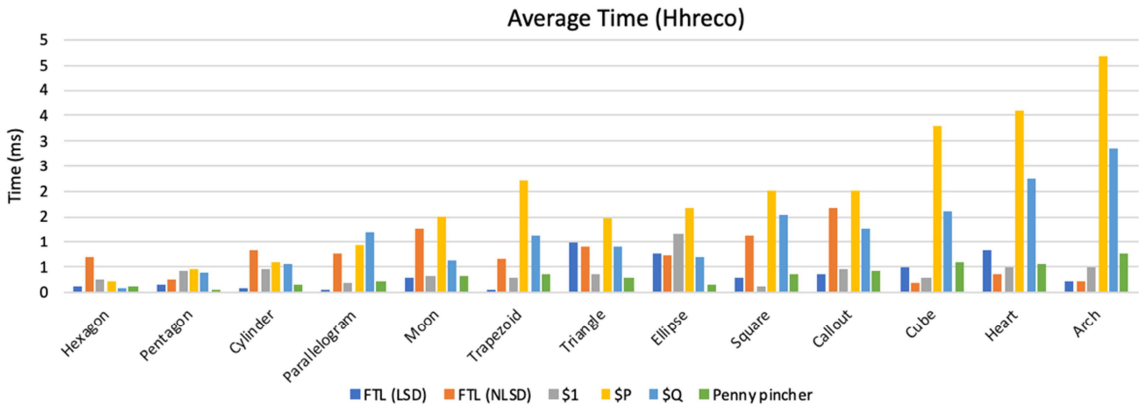

Fig. 8 Average recognition time resulting from the HHreco dataset

Utopiano recognition time analysis: Figure 9 shows the results of the average recognition time of the *Utopiano* dataset. It is indicated that *!FTL(NLSD)* algorithms, followed by Penny Pincher, are the fastest to recognize *NicIcon* gestures. The fastest representations to be recognized were the letters F and K.

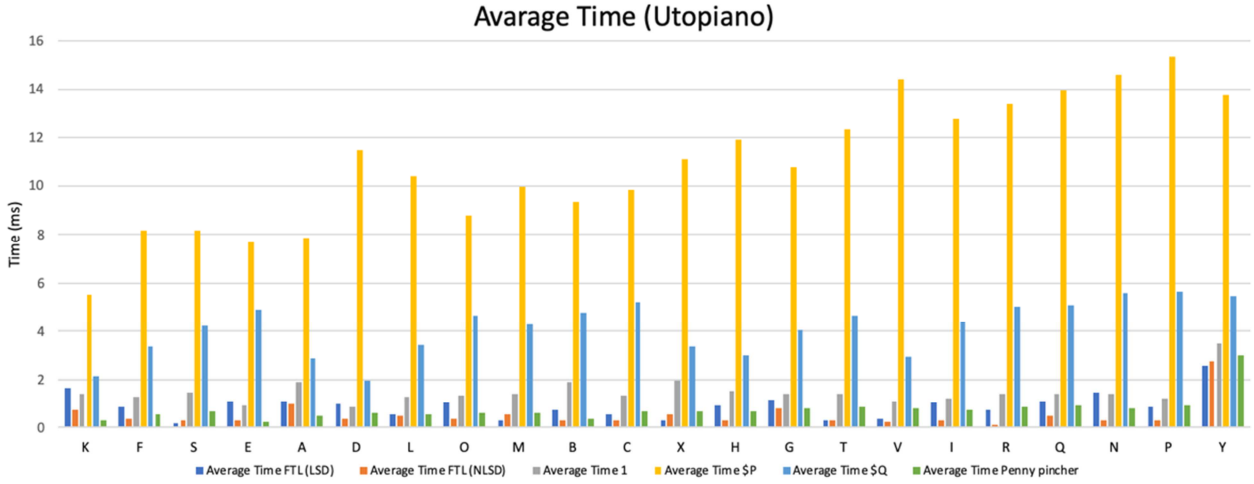

Fig. 9 Average recognition time resulting from the *Utopiano* dataset

Table 5 shows a compilation of the results obtained from the comparative evaluation presented in previous section. The table indicates the two best algorithms, according to the recognition rate and the recognition time for each of the datasets.

TABLE V
SUMMARY OF THE RESULTS (EXPRESSED IN PERCENTAGE VALUES)

Datasets Measures	***NicIcon***			***HHreco***			***Utopiano***		
Recognition rate	Position	Recognizer	Score	Position	Recognizer	Score	Position	Recognizer	Score
	1st	!FTL(LSD)	9.79%	1st	!FTL(NLSD)	25.26%	1st	$1	47.27%
	2nd	!FTL(NLSD)	9.47%	2nd	$1	18.94%	2nd	!FTL(NLSD)	26.27%
Recognition time	Position	Recognizer	Score	Position	Recognizer	Score	Position	Recognizer	Score
	1st	Penny Pincher	0.216ms	1st	Penny Pincher	0.34ms	1st	!FTL(NLSD)	0.518ms
	2nd	!FTL(NLSD)	0.372ms	2nd	!FTL(LSD)	0.361ms	2nd	Penny Pincher	0.757ms

In the first dataset, *NicIcon*, *!FTL(LSD)* is distinguished as the algorithm with the highest recognition rate with 9.79%. The next algorithm is *!FTL(NLSD)* with a 9.47% recognition rate, that is, only 0.32% lower than *!FTL(LSD)*. In the case of Electricity and Roadblock, the two gestures with the greatest number of successes, the algorithms did not meet the expectations, since it was proposed to have only an error rate of 21.43%, in both gestures, and a 90.21% and 90.53% error with !FTL(NLSD), while !FTL (LSD) showed 71.67% and 85.94% of failed acknowledgments. Likewise, the Penny Pincher algorithm is the fastest for the *NicIcon* dataset with 0.22 ms, but its recognition rate is only 5.93%. The next fastest is *!FTL(NLSD)* with 0.37 ms, that is 0.15 ms more. Because of its recognition rate and speed, it makes the *! FTL (NLSD)* algorithm the most efficient for the *NicIcon* dataset.

In *HHreco*, we can see that *!FTL(NLSD)* as the algorithm with the best recognition rate with 25.26 %. The algorithm that obtained second place was $1 with an 18.94% recognition rate, that is, it is 6.32% lower than! FTL (NLSD). In the case of Ellipse and Arch, which are the two most successful gestures, the proposal was complied with, since Ellipse with *!FTL(NLSD)* had 95.29% assertiveness, while with Arch he obtained 5.10% of hits over 15.34% of the proposed error rate. On the other hand, with $1 and Ellipse it resulted in 15.10% of assertiveness over the estimated 70%, and with Arch it achieved 28.63%, exceeding the indicated error rate of 15.34%. From the point of view of recognition time, the Penny Pincher algorithm was the fastest for the *HHreco* dataset with 0.34 ms, only that its recognition rate only reached 11.66%. The next fastest algorithm was *!FTL(NLSD)* with 0.36 ms, that is 0.02 ms more than the Penny Pincher time. This demonstrates its efficiency again.

Finally, in the *Utopiano* Alphabet dataset, it is shown that the *$1* algorithm has the best recognition rate with 47.27%. The second algorithm with the highest number of hits was *!FTL(NLSD)* with 26.27%, that is, it is 21% less than *$1*. In the case of the letter P and N, which are the two gestures with the highest recognition rate, the letter P with $ 1 had 74% of assertiveness, while with the letter N it got 67% of hits on the 18.18% of the error rate that was defined in both letters. *!FTL(NLSD)* and the letter P obtained 19% over the 18.18% that was estimated to be correct, and with the letter N it achieved 35% of assertiveness, exceeding the error rate indicated above. Regarding the recognition time, the *!FTL(NLSD)* algorithm was the fastest in the *Utopiano* dataset with 0.52 ms, followed by Penny Pincher with 0.77 ms, that is, only a difference of 0.25 ms. And for the third time, *!FTL(NLSD)* was once again the fastest and one of the most accurate algorithms.

IV. CONCLUSION

This paper presented a comparative analysis of the algorithmic efficiency of different algorithms on two existing datasets and a dataset generated for this purpose. Although the algorithms were evaluated under the same experimental conditions, i.e., with the same data sets on the same computer, the recognition rates are lower than expected. Although the experiment was performed on the same computer, there were peaks of times that varied due to the events that the browser decided to allocate more time to perform. For this reason, it is recommended to make the time allocation for each process more uniform. One proposal is to implement a service with low-level language such as C, to perform the evaluation of the algorithms within a microcontroller.

The evaluation process was semi-automatic since the candidate gestures had to be done manually for each dataset and compared with the dataset gestures. Therefore, it is suggested to automate the evaluation [20] that can be taken as a reference a gesture of the dataset randomly and evaluate the remaining gestures against this selected gesture. This process would be repeated for each gesture in the dataset: while there was an unassessed gesture, it would be selected and evaluated against the remaining gesture group. With this form of evaluation, candidate gestures by the user would be avoided. In the same line of automatic evaluation, gestures stored in the experiment environment could be subject to evaluation of design guidelines in terms of articulation, including their operationalization in HTML (as in [21]) for instance.

One of the main drawbacks encountered in making the comparative analysis has to do with the different formats for data that exist by dataset. Each dataset has a different way of saving the repetitions of the gestures, in other words, the way to store the name of the repetition that was performed, the way to display the coordinates of the points and how each file is saved with all repetitions according to user or representation. Therefore, it is suggested that a protocol be made for algorithms to run in similar environments.

The use of datasets helped save gesture diversification time for the evaluation process, and as mentioned earlier, there are several datasets that could not be used in this project. For this reason, it is recommended to collect and use other datasets in the future. There are datasets that have characteristics that challenge recognition algorithms, such as datasets with 3D figures. This opens the opportunity to evaluate more algorithms that specialize in these gestures, as is the case of $3, which can recognize figures in third dimension.

REFERENCES

[1] Saffer, D.: Designing Gestural Interfaces: Touchscreens and Interactive Devices. O'Reilly Media, Inc. (2009)

[2] Zhai, S., Kristensson, P., Appert, C., Andersen, T., Cao, X.: Foundational issues in touch-surface stroke gesture design: An integrative review. Found. Trends Human-Computer Interaction. 5(2), 97–205 (Feb 2012). https://doi.org/10.1561/1100000012.

[3] Schipor, O.A., Vatavu, R.D., Vanderdonckt, J.: Euphoria: A scalable, event-driven architecture for designing interactions across heterogeneous devices in smart environments. Information and Software Technology 109, 43–59 (2019). https://doi.org/https://doi.org/10.1016/j.infsof.2019.01.006.

[4] Simarro, F.M., Lopez-Jaquero, V., Vanderdonckt, J., González, P., Lozano, M.D., Limbourg, Q.: Solving the mapping problem in user interface design by seamless integration in idealxml. In: Gilroy, S.W., Harrison, M.D. (eds.) Interactive Systems, Design, Specification, and Verification, 12th International Workshop, DSVIS 2005, Newcastle upon Tyne, UK, July 13-15, 2005, Revised Papers. Lecture Notes in Computer Science, vol. 3941, pp. 161–172. Springer (2005). https://doi.org/10.1007/11752707_14.

[5] Wobbrock, J.O., Aung, H.H., Rothrock, B., Myers, B.A.: Maximizing the guessability of symbolic input. In: CHI'05 Extended Abstracts on Human Factors in Computing Systems. pp. 1869–1872. CHI EA'05, ACM, New York, NY, USA (2005). https://doi.org/10.1145/1056808.1057043.

[6] Gheran, B.F., Vanderdonckt, J., Vatavu, R.D.: Gestures for smart rings: Empirical results, insights, and design implications. In: Proceedings of the 2018 Designing Interactive Systems Conference. pp. 623–635. DIS'18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3196709.3196741.

[7] Wobbrock, J.O., Wilson, A.D., Li, Y.: Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology. pp. 159–168. UIST'07, ACM, New York, NY, USA (2007). https://doi.org/10.1145/1294211.1294238.

[8] Vatavu, R.D., Anthony, L., Wobbrock, J.O.: Gestures as point clouds: A $p recognizer for user interface prototypes. In: Proceedings of the 14th ACM International Conference on Multimodal Interaction. pp. 273–280. ICMI '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2388676.2388732.

[9] Vatavu, R.D., Anthony, L., Wobbrock, J.: $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. pp. 623–635. MobileHCI'18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3229434.3229465.

[10] Vanderdonckt, J., Roselli, P., Pérez-Medina, J.L.: 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. pp. 125–134. ICMI'18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3242969.3243032.

[11] Taranta, II, E.M., LaViola, Jr., J.J.: Penny pincher: A blazing fast, highly accurate $-family recognizer. In: Proceedings of the 41st Graphics Interface Conference. pp. 195–202. GI'15, Canadian Information Processing Society, Toronto, Ont., Canada, Canada (2015), http://dl.acm.org/citation.cfm?id=2788890.2788925

[12] Niels, R., Willems, D., Vuurpijl, L.: The NicIcon database of handwritten icons for crisis management. Nijmegen Institute for Cognition and Information Radboud University Nijmegen, Nijmegen, The Netherlands 2 (2008)

[13] Hse, H., Newton, A.R.: Graphic Symbol Recognition Toolkit (HHreco) Tutorial. Department of Electrical Engineering and Computer Sciences, University of California at Berkeley (2003)

[14] Hse, H., Newton, A.R.: Sketched symbol recognition using zernike moments. In: Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. vol. 1, pp. 367–370. IEEE (2004)

[15] Ager, S.: Omniglot: The Online Encyclopedia of Writing Systems & Languages. Simon Ager (1998)

[16] Willems, D., Niels, R., van Gerven, M., Vuurpijl, L.: Iconic and multi-stroke gesture recognition. Pattern Recognition 42(12), 3303 – 3312 (2009). https://doi.org/https://doi.org/10.1016/j.patcog.2009.01.030.

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

[18] Coyette, A., Schimke, S., Vanderdonckt, J., Vielhauer, C.: Trainable sketch recognizer for graphical user interface design. In: Baranauskas, C., Palanque, P., Abascal, J., Barbosa, S.D.J. (eds.) Human-Computer Interaction – INTERACT 2007. pp. 124–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[19] Signer, B., Kurmann, U., Norrie, M.: igesture: A general gesture recognition framework. In: Ninth International Conference on Document Analysis and Recognition (ICDAR 2007). vol. 2, pp. 954–958 (Sep 2007). https://doi.org/10.1109/ICDAR.2007.4377056

[20] Vanderdonckt, J., Beirekdar, A.: Automated web evaluation by guideline review. Journal of Web Engineering 4(2), 102–117 (2005), http://www.rintonpress.com/xjwe4/jwe-4-2/102-117.pdf

[21] Beirekdar, A., Vanderdonckt, J., Noirhomme-Fraiture, M.: A framework and a language for usability automatic evaluation of web sites by static analysis of HTML source code. In: Kolski, C., Vanderdonckt, J. (eds.) Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France. pp. 337–348. Kluwer (2002)