$N-Protractor: A Fast and Accurate Multistroke Recognizer

Lisa Anthony[†] and Jacob O. Wobbrock[‡]

[†]University of Maryland Baltimore County, [‡]The Information School, DUB Group, University of Washington

ABSTRACT

Prior work introduced $N, a simple multistroke gesture recognizer based on template matching, intended to be easy to port to new platforms for rapid prototyping, and derived from the unistroke $1 recognizer. $N uses an iterative search method to find the optimal angular alignment between two gesture templates, like $1 before it. Since then, *Protractor* has been introduced, a unistroke pen and finger gesture recognition algorithm also based on template-matching and $1, but using a closed-form template-matching method instead of an iterative search method, considerably improving recognition speed over $1. This paper presents work to streamline $N with Protractor by using Protractor's closed-form matching approach, and demonstrates that similar speed benefits occur for multistroke gestures from datasets from multiple domains. We find that the Protractor enhancements are over 91% faster than the original $N, and negligibly less accurate (<0.2%). We also discuss the impact that the number of templates, the input speed, and input method (*e.g.*, pen *vs.* finger) have on recognition accuracy, and examine the most confusable gestures.

KEYWORDS: Multistroke gesture recognition, stroke recognition, template matching, $N, Protractor, evaluation.

INDEX TERMS: H.5.2. [Information interfaces and presentation]: User interfaces—input devices and strategies; I.5.5. [Pattern recognition]: Implementation—interactive systems.

1 INTRODUCTION

Pen and finger gestures are becoming ever more important to user interfaces. Integrating application-specific gestures, such as special commands or sketch or handwriting input, requires user interface prototypers to know much about gesture recognition in order to choose the right recognizer. In addition to performance, considerations such as ease of integration and simplicity of training are important in deciding which approach to take. Although gesture recognition is becoming more common, sketch-based input still does not enjoy mainstream support on new platforms, requiring UI prototypers to "grow their own" or port tools to their new device.

$N is a multistroke pen and finger gesture recognition algorithm [1], a simple, easy-to-train geometric template matcher based on the $1 unistroke recognizer [5]. Targeted to be easy to port to new platforms by virtue of its straightforward, geometry-based algorithm, $N has experienced swift uptake in rapid prototyping for interactive systems. Implementations in JavaScript, C# and Objective-C already exist, and an iPhone app has been released that uses $N to accept touch gesture input. The clarity of $N enables such swift uptake; for example, the C# version consists of just 240 lines of code and uses only basic geometry computations. See the $N webpage[1] for further details, including pseudo-code and open-source implementations in JavaScript and C#.

A major limitation of $N (and $1 before it) is the computational demand of the method used to find the optimal angular alignment between two gestures. $N iteratively rotates a candidate gesture by some number of degrees using the *Golden Section Search* algorithm (GSS) [4] (pp. 397-402), to determine the best angular alignment with any given template. Then the Euclidean distance between points in the rotated candidate gesture and the template gesture defines the quality of the match. An extension of $1 called *Protractor* [3] has been introduced which eliminates this iterative search for the best angular alignment by using a closed-form approach based on inverse cosine distances. Protractor has been shown to significantly improve speed of recognition over $1 [3].

Therefore, we have incorporated the matching method used in unistroke Protractor into $N to determine whether the same speed improvements materialize in the multistroke formulation of this approach. We find that performance results comparing the original $N ($N-GSS) to the Protractor-enhanced $N ($N-Protractor) on the same datasets show speed benefits of over 91% without penalizing complexity or accuracy significantly.

2 PRIOR WORK

A brief discussion of how both $N and Protractor work is provided here, but see their original papers for full details of implementation and previous evaluations.

2.1 $N Multistroke Recognizer

Details of the $N recognizer, including a complete pseudocode listing, can be found in the original paper [1]. The multistroke $N recognizer is based on the $1 unistroke recognizer [5]. Both recognizers use a geometric template matching approach, comparing new *candidate* gestures to loaded *templates* by iteratively searching for the optimal angular alignment between two gestures and comparing distances between corresponding points. Both candidates and templates are pre-processed using the same steps to standardize the gestures before alignment is performed. $N goes beyond the original $1 recognizer by supporting multistroke gestures by sampling gestures "through the air", *i.e.*, during the pen-up part of the multistroke gesture. To remain robust to stroke orders and stroke directions, $N automatically computes all possible permutations of a multistroke, enabling it to recognize a gesture made with a different stroke order or stroke direction than the loaded templates [1].

2.2 Protractor

Li has published Protractor [3], an extension to the original $1 recognizer that uses a closed-form solution to find the optimal angular alignment between a template and a candidate gesture. Protractor's approach significantly reduces the computation needed during the matching process by removing the iterative search over angles. Like $1, Protractor computes the similarity between a candidate and template using a distance metric.

[†]Baltimore, MD, USA, [‡]Seattle, WA, USA
[†]lanthony@umbc.edu, [‡]wobbrock@uw.edu

[1] http://depts.washington.edu/aimgroup/proj/dollar/ndollar.html

Protractor's distance metric finds the angle between two vector-based representations of unistroke gestures in an n-dimensional space (Protractor uses 16 dimensions by resampling gestures to 16 points). Before computing this distance, however, it is important to ensure the two gestures are optimally aligned so that the minimum possible distance can be found. To do so, Protractor calculates the inverse minimum cosine distance between two gestures, a closed-form operation rather than the iterative one used by $1 and $N. Pseudo-code is provided on the Protractor website[2]. Li [3] reported significant speed improvements over $1.

3 DATASETS

We evaluated the Protractor-enhanced version of $N on three datasets. The Mixed Multistroke Gesture (MMG) dataset is a new dataset collected for this paper. The $1 unistroke dataset was previously collected and used to evaluate the original versions of $1 and of Protractor, and the algebra dataset was previously collected and used to evaluate the original version of $N.

3.1 Mixed Multistroke Gesture Dataset

For this paper, we have collected a new multistroke pen gesture dataset called "Mixed Multistroke Gestures" (MMG) that contains a set of symbols representative of those used in multistroke gesture input applications. The dataset is a mixed multistroke and unistroke dataset consisting of 3200 samples drawn by 20 different users on a Windows-based Tablet PC, including 7 females and 13 males, ranging in age from 18 to 33 years. Due to the rise in popularity of touch-based gesture interfaces on smartphones like the iPhone and Android platform, more than half of the users were familiar with touch and finger input on digital devices; few were as familiar with pen or stylus input but most had used pens or styli before providing samples for us.

Each user wrote 10 samples per symbol; half of the users entered data via their index finger and half of the users entered data via the digital stylus. The symbol set, originally defined in [1], includes the following 16 symbols: {arrowhead, asterisk, D, exclamation point, five-pointed star, H, half-note, I, line, N, null symbol, P, pitchfork, six-pointed star, T, X}. See Figure 1 for the symbol set used in this dataset; the number in parentheses next to each symbol name is the target number of strokes we asked users to enter. The dataset consists of 87% multistroke and 13% unistroke samples. The maximum number of strokes across all samples is 5. Note that the symbols were shown to the user with the number of strokes indicated in Figure 1, and the maximum number of strokes is 3. Therefore, samples with strokes numbers higher than 3 are technically mis-entered data, but this occurred in fewer than 0.6% of the samples.

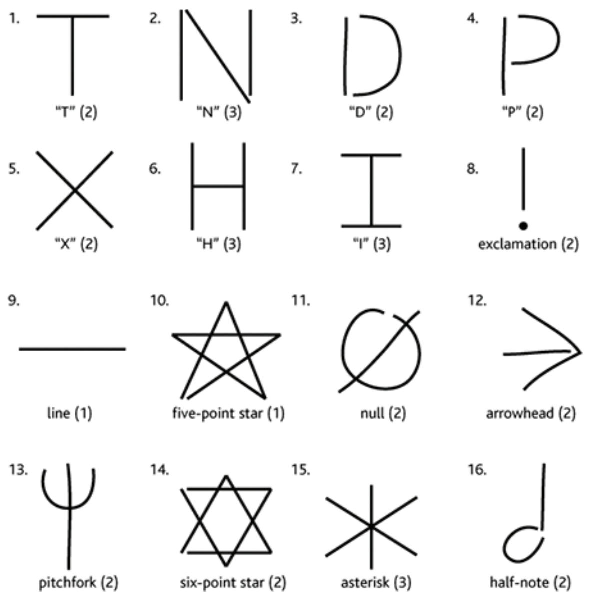

Figure 1: Mixed Multistroke Gesture (MMG) symbol set, using symbols defined in [1].

[2] http://yangl.org/protractor/protractor.pdf

3.2 $1 Unistroke Dataset

$1 was originally evaluated on a dataset similar to MMG that only included unistroke symbols, and was drawn by adults [5]. Protractor was also evaluated on this dataset [3]. In addition to the new MMG dataset, we were able to evaluate our new $N-Protractor on this benchmark dataset ("Unistrokes").

3.3 Algebra Dataset

$N was originally evaluated on a challenging dataset of unconstrained algebra symbols drawn by middle and high school students to test its limits [1]. In addition to the other two datasets mentioned, we were able to evaluate $N-Protractor on this real-world dataset ("Algebra").

4 BENCHMARK TESTS

A separate writer-dependent benchmark test was run for each recognition approach and dataset in this paper. The procedure used for evaluation, which we call "random-100," mirrored that described in previous work [1,5]. The $N recognizer was configured with the parameters that yielded the highest performance in the original evaluation reported on the algebra corpus [1]. Future evaluations might vary the parameters used to determine to what extent they are domain-dependent, if at all. Two versions of $N were tested on each dataset, one using the closed-form Protractor method ($N-Protractor) and one using the original iterative method ($N-GSS).

5 ANALYSIS AND RESULTS

We present results of the evaluation in terms of recognition accuracy and recognition speed for all three datasets. We also report findings on the impact of input method (*e.g.*, using one's finger *vs.* a digital stylus) and the impact of gesture articulation speed (*e.g.*, being more or less careful when entering gestures) on $N-Protractor's performance for the MMG dataset. We also report the most highly confusable symbols in the MMG dataset.

5.1 Recognition Accuracy

In the original evaluation of $N-GSS [1], accuracy levels of 96.6% were achieved with 15 templates per symbol on the Algebra dataset, and 96.7% with 9 templates per symbol on the Unistrokes dataset. We replicated the $N-GSS tests, and achieved about the same or better results: 95.4% accuracy with 15 templates per symbol on the Algebra dataset, and 97.7% accuracy with 2 templates per symbol on the Unistrokes dataset (accuracy was higher on Unistrokes even when treating unistrokes as multistrokes and storing both directional permutations). On the new MMG dataset, accuracy was approximately 97% for both versions of $N with just 4 templates per symbol loaded. Although the difference is extremely small, the $N-Protractor is consistently and statistically significantly less accurate on the MMG dataset as the number of training examples per symbol increases ($t(8)=5.17$, $p<0.05$). In practice, this difference is so small that it likely does not impact real-world use. The tiny performance hit experienced with the Protractor matching method is far outweighed by the speed benefits. When using the Protractor matching method

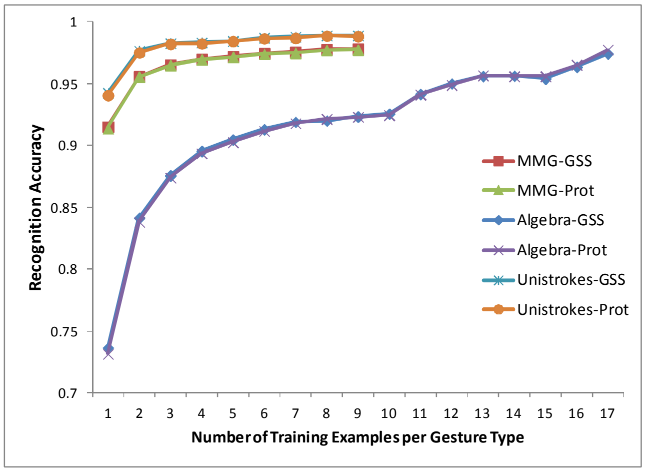

Figure 2: $N recognition accuracy per dataset when using the Protractor matching method *vs.* the GSS matching method as the number of training examples increases.

instead of GSS, accuracy stayed about the same, but as discussed in the next section, recognition speed increased more than fivefold. Figure 2 shows the recognition accuracy performance of $N-GSS and $N-Protractor on each dataset for each level of templates per symbol.

5.2 Recognition Speed

The time taken for our evaluation was far less with $N-Protractor than with $N-GSS. For the MMG dataset, $N-Protractor took 18.70 minutes to complete on a Dell Studio 1558 laptop running 64-bit Windows 7 with a 2.27 GHz Intel Core i5 CPU and 4.00Gb RAM. $N-GSS took 208.95 minutes to complete the same evaluation on the same data on the same computer. This result represents a more than 91% time savings, and is the result of the closed form matching method used in Protractor. On the Algebra dataset, $N-Protractor took 10.62 minutes to complete on the same machine as the earlier tests were run, whereas $N-GSS took 59.29 minutes to complete the same evaluation on the same data on the same computer, an 82% savings. On the Unistrokes dataset, $N-Protractor took 1.16 minutes to complete, whereas $N-GSS took 5.83 minutes, an 80% savings. Thus, the time savings overall varied per dataset, but was dramatic in all cases.

Not only did the overall time decrease considerably with Protractor, but the time per recognition increased less steeply per additional template with the Protractor enhancement than it did with the GSS matching method. In both cases, time to recognize a gesture increases as the number of training examples per gesture type increases, because the recognizer is comparing new candidates to iteratively more possible templates. However, with Protractor, this added cost is much less than with GSS. Figure 3 shows the increase in time in milliseconds to recognize a gesture with $N-GSS and $N-Protractor on each dataset as the number of training examples per gesture type increases. With 9 training examples per gesture type, GSS took 25.8 ms to recognize a gesture in the MMG dataset, whereas Protractor took only 1.9 ms to do so. On the Algebra dataset with the same number of templates loaded, it took only 6.4 ms to recognize a gesture with GSS, and only 0.67 ms with Protractor. On the Unistrokes dataset, it took 3.89 ms to recognize a gesture with GSS and only 0.42 ms with Protractor. For all datasets, this difference in time taken per recognition as the number of training examples increases was statistically significant in favor of Protractor (MMG: ($t(8)=5.48$, $p<0.05$); Algebra: $t(16)=7.97$, $p<0.05$; Unistrokes: $t(8)=5.49$, $p<0.05$). The degree of time savings per recognition varied per

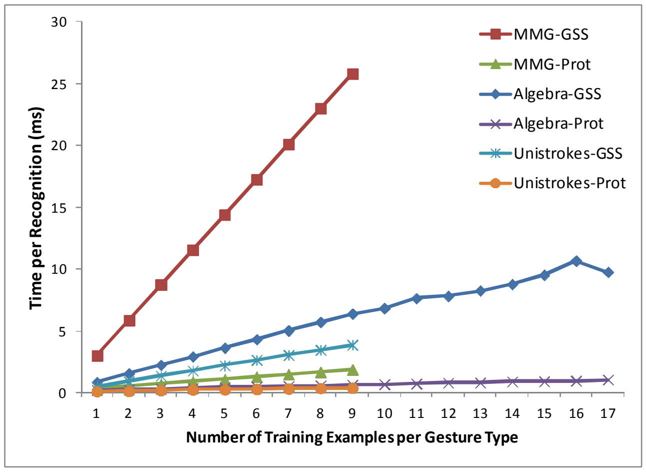

Figure 3: $N recognition speed per dataset when using the Protractor matching method *vs.* the GSS matching method as the number of training examples increases.

dataset, but remained a dramatic and significant improvement when using Protractor.

The original Protractor paper [3] described a similar speed gain, taking less than 0.5 ms per gesture while $1 (using GSS) took over 3 ms on the Unistrokes dataset. The time per gesture was less in that evaluation than in this one because only unistroke gestures were tested, and $N-Protractor (as well as $N-GSS) represents each multistroke gesture evaluated here as multiple unistrokes, increasing the number of comparisons that are done.

5.3 Impact of Input Method

We also investigated whether the use of one's finger or a digital stylus yields better recognition accuracy with $N. We analyzed the performance of $N-Protractor in relation to input method (*e.g.*, finger or stylus) for the MMG dataset. We found that $N-Protractor was significantly more accurate with one's finger than with the stylus ($t(8)=6.38$, $p<0.05$), shown in Figure 4. This effect could be due to users' higher comfort with finger gestures *vs.* stylus gestures, or due to increased jitter while holding the stylus.

5.4 Impact of Input Speed

During data collection for MMG, users were asked to enter gesture samples at three different speeds: slow, medium, and fast. We used the same definitions of the speeds as in the original $1

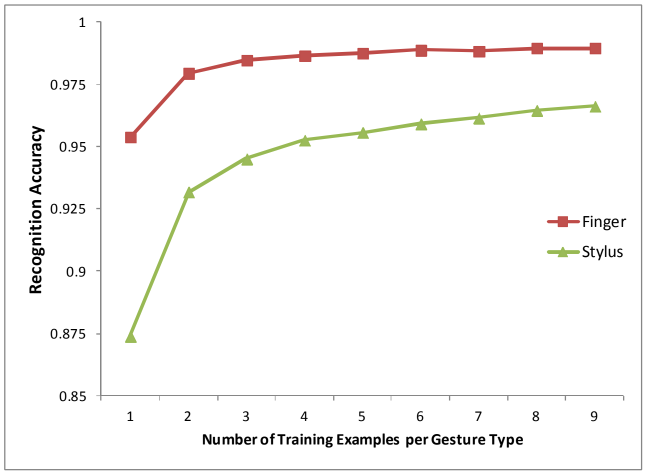

Figure 4: $N-Protractor recognition accuracy on the MMG dataset on gestures entered via a finger *vs.* a digital stylus.

paper [5]: for slow gestures, users were asked to "be as accurate as possible;" for medium gestures, users were asked to "balance speed and accuracy;" for fast gestures, users were asked to "go as fast as you can." These messages appeared before each set of gestures at the new speed. In a repeated measures ANOVA on input speed controlling for user, $N-Protractor showed no significant difference in recognition accuracy at the three different speeds ($F(2,38)=1.72$, *n.s.*) for the MMG dataset, indicating that it is suitable for a wide variety of contexts in which users may be more careful or less careful when entering gesture input.

5.5 Most Confusable Gestures

For the MMG dataset, there was a significant difference in how well specific symbols were recognized by $N-Protractor ($F(1,15)=3054.9$, $p<0.05$). The least well-recognized symbol was the exclamation point, with much lower accuracy (87% correct) than the next highest symbol (six-point star, 93% correct), likely due to inconsistencies in how users made the gesture. The most well-recognized symbols, in decreasing order, were the pitchfork, asterisk, and "I" at roughly 98% correct each. Predictably, which gestures were harder to recognize did not change depending on which matching method, GSS or Protractor, was used.

The most highly confused pairs (cases in which the mis-recognition count was over 538, or 1% of tests per character) are given in Table 1. Although never more than 5% of tests per symbol were incorrect, from these results, we can conclude that $N-Protractor has difficulty recognizing gestures with very small or short component strokes (*e.g.*, often confusing exclamation points for half notes). This difficulty occurs despite the scaling done during pre-processing, which may exaggerate small jitters or abnormalities in the short strokes. $N-Protractor also tends to have difficulty with 1-D gestures such as lines and exclamation points. Because no pair of gestures tended to be confused for each other (*e.g.*, exclamation points were confused for half notes, but half notes were not confused for exclamation points), there does not seem to be a specific conclusion we can draw about the types of symbols $N-Protractor has trouble discerning between.

Table 1: Gestures in the MMG dataset most highly confused by $N-Protractor. Total number of each confusion type is given, as well as the percentage of tests (out of 53800 per tested gesture) that were confused in the given way.

Tested gesture	Confused gesture	No. of confusions	% tests confused
Exclamation point	Half note	2488	4.6%
Line	[no result]	1466	2.7%
Exclamation point	N	1162	2.2%
H	N	822	1.5%
Six point star	Null	777	1.4%
Five point star	[no result]	749	1.4%
Exclamation point	Arrow	737	1.4%
Exclamation point	T	703	1.3%
P	D	608	1.1%
Half note	[no result]	561	1.0%
N	P	553	1.0%

6 Conclusion

We have presented an extension of the popular $N multistroke pen and finger gesture recognizer to use the newer Protractor matching method previously applied to speed up $1 for unistroke gestures. We have shown that, both on a newly collected mixed multistroke gestures (MMG) dataset and on previously benchmarked datasets in different domains (multistroke Algebra and Unistrokes), Protractor yields the same speed benefits for $N as it did for $1, with negligible accuracy cost. Furthermore, we have explored impact of input method and speed on $N-Protractor's recognition accuracy for the MMG dataset, and found that, while haste / carefulness had no effect, using one's finger rather than a stylus tended to be more accurate with $N-Protractor. Finally, we presented confusion results for $N-Protractor's performance on the MMG dataset and some preliminary conclusions about the suitability of $N-Protractor for symbols of different types.

7 Future Work

While this paper has presented an optimization for $N called $N-Protractor to alleviate the time cost of representing multistrokes as unistroke permutations, an important limitation still exists for $N in terms of the space cost. A gesture such as a cube may be drawn with up to 9 strokes, which would involve 185 million permutations and is infeasible to store on modern desktops, let alone mobile devices. Future work will involve adapting $N-Protractor for use with symbols of many component strokes. We intend to explore alternative data structures, representations or storage strategies (*e.g.*, generating the permutations on the fly rather than storing them) for $N-Protractor that can retain its stroke-order and stroke-direction independence without representing all possible permutations explicitly. Alternatively, another extension to $1 to handle multistrokes besides $N has been proposed [2] in which multistroke permutations are not generated. In order to retain robustness to stroke order and direction, a complex simulated annealing method is used to iteratively find the minimum sum of the distances between all pairs of points to find the best possible match, which determines the score for that template. We do not believe this matching method is appropriate for the intended rapid prototyping use cases of $N, but perhaps a combination of this approach and $N-Protractor can yield better results while retaining simplicity and ease of use. In addition, we continue to study $N's suitability for non-pattern matching and sketch and gesture algorithm experts by exploring adoption, ease of use, and simplicity to port $N to new platforms.

Acknowledgments

The authors thank Peng Chu for assistance with collecting the MMG dataset, and the users who provided the MMG samples.

References

[1] L. Anthony and J.O. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. *Proceedings of Graphics Interface '10* (Ottawa, Canada, May 31-June 2, 2010), 245-252. Canadian Information Processing Society, 2010.

[2] M. Field, S. Gordon, E. Peterson, R. Robinson, T. Stahovich, and C. Alvarado. The effect of task on classification accuracy: Using gesture recognition techniques in free-sketch recognition. *Proceedings of Sketch-Based Interfaces and Modeling '09* (New Orleans, Louisiana, August 1-2, 2009), 499-512. ACM Press, 2009.

[3] Y. Li. Protractor: a fast and accurate gesture recognizer. *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems '10* (Atlanta, Georgia, April 10-15, 2010), 2169-2172. ACM Press, 2010.

[4] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, 1992.

[5] J.O. Wobbrock, A.D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. *Proceedings of ACM Symposium on User Interface Software and Technology '07* (Newport, Rhode Island, October 7-10, 2007), 159-168. ACM Press, 2007.