# Penny Pincher: A Blazing Fast, Highly Accurate $-Family Recognizer

Eugene M. Taranta II [*]
Department of EECS
University of Central Florida
4000 Central Florida Blvd.
Orlando, FL, 32816

Joseph J. LaViola Jr. [†]
Department of EECS
University of Central Florida
4000 Central Florida Blvd.
Orlando, FL, 32816

## Abstract

The $-family of recognizers ($1, Protractor $N, $P, 1¢, and variants) are an easy to understand, easy to implement, accurate set of gesture recognizers designed for non-experts and rapid prototyping. They use template matching to classify candidate gestures and as the number of available templates increase, so do their accuracies. This, of course, is at the cost of higher latencies, which can be prohibitive in certain cases. Our recognizer *Penny Pincher* achieves high accuracy by being able to process a large number of templates in a short amount of time. If, for example, a recognition task is given a 50$\mu$s budget to complete its work, a fast recognizer that can process more templates within this constraint can potentially outperform its rival recognizers. Penny Pincher achieves this goal by reducing the template matching process to merely addition and multiplication, by avoiding translation, scaling, and rotation; and by avoiding calls to expensive geometric functions. Despite Penny Pincher's deceptive simplicity, our recognizer, with a limited number of templates, still performs remarkably well. In an evaluation compared against four other $-family recognizers, in three of our six datasets, Penny Pincher achieves the highest accuracy of all recognizers reaching 97.5%, 99.8%, and 99.9% user independent recognition accuracy, while remaining competitive with the three remaining datasets. Further, when a time constraint is imposed, our recognizer always exhibits the highest accuracy, realizing a reduction in recognition error of between 83% to 99% in most cases.

**Keywords:** Gesture recognition, $-family, template matching.

**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

As pen- and touch-based interfaces become ubiquitous via the widespread use of smart phones, pocket computers, and tablet computers, the need for high quality gesture recognition also continues to rise. Enabling technologies such as the iTunes Store, Google Play, and Windows Store makes it possible for individual developers to create and easily distribute small to large scale software for these devices, which has attracted numerous professionals as well as casual hobbyists. However, without expert knowledge in pattern recognition, a developer is unable to create even the simplest custom gesture-based interface for his or her software. Wobbrock et al. [17] began to address this issue in their seminal work when they introduced the $1 recognizer. This is a straightforward, highly accessible, and easy to implement recognizer that spurred a flurry of research known as the $-family of recognizers [2, 3, 6, 9, 16, 17], their common theme of which is simplicity.

[*]e-mail: etaranta@gmail.com
[†]e-mail: jjl@eecs.ucf.edu

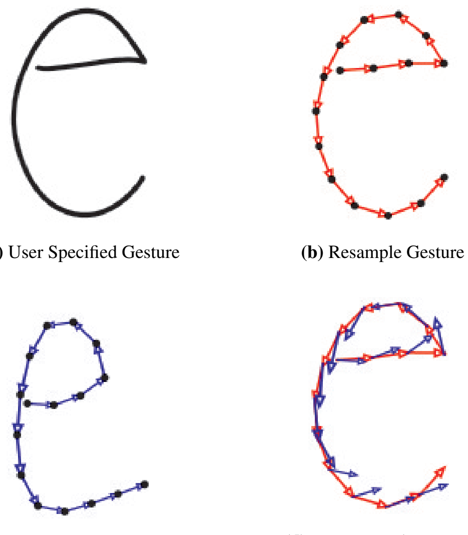
(a) User Specified Gesture
(b) Resample Gesture
(c) Best Match
(d) Between-point Vectors Comparison

Figure 1: A candidate gesture (a) is given by the user, which is then resampled into a set of equidistance segments (b). Given a template gesture (c) we compare the corresponding between-point vectors to find the best match. In this case, the 'e' template is the best match because it minimizes the sum of the angles of the corresponding resampled input and template vectors as compared to all other templates.

The $-family exhibits several benefits, the most obvious of those being their all around straightforwardness, as already stated. This makes them ideal not only for rapid prototyping, but because they are also generally accurate, they are a good choice for situations where existing gesture recognition libraries cannot be used or when licensing constraints make their use prohibitive. Because they are quick to implement, $-family recognizers are also ideal for student studies in gesture-based user interfaces. Another argument for their use, provided by Li [9], is that for custom gestures, users are unwilling to provide more than a small number of training examples, which makes statistical based methods like Rubine's [13] linear discriminator impractical. Further, each recognizer is designed to solve a different problem. For example, $1 was designed for unistroke gestures, $N for multistroke, $P for memory reduction, Protractor to overcome the speed impediments of $1, 1¢ for fast recognition of rotation invariant gestures, and so on. However, all of these varieties rely on template matching and as such, tend to improve in accuracy as the number of training templates increase.

In this work we address the problem of how to create a recognizer that is accurate within a time constraint. For instance, suppose that the recognizer is part of a game. The recognizer then has to share computational resources with several other components including AI, the rendering engine, animation system, physics simulator, event and message handlers, networking, scripts, and so on

[5]. Therefore, a software architect may allocate only 50 microseconds to the gesture recognition task when a new stroke is collected, to ensure that the process is not disruptive to other game tasks. This poses two issues. First, with such a tight constraint, certain recognizers cannot be used because they are simply too slow. Second, this limits the number of templates a recognizer can check during the matching process. Another concern is that recognition latency plays a critical role in user perception and interface design. Gray and Boeham-Davis [4], for instance, have shown that even seemingly negligible latencies (on the order of milliseconds) can influence how users interact with software. Therefore, to address these issues we introduce *Penny Pincher*, an accurate recognizer that achieves high accuracy through speed. This $-family extension is designed to do the absolute minimum amount of work possible to match candidate gestures with templates so that more templates can be evaluated within the same amount of time as other recognizers. As a result, our recognizer is significantly more accurate than its kindred.

Most other recognizers use Euclidean distance to compare gestures whereas we accomplish our goal by comparing between-point vectors, see Figure 1. As do other recognizers, we resample the input candidate stroke to a constant number of points; however, we avoid rotating, scaling, and translating the gesture. The only limitation of this choice is that our recognizer is not rotation invariant, but with enough templates loaded, this is not an issue. Further, after resampling the input, we use only addition and multiplication operations; that is, we also avoid any calls to computationally expensive geometric functions, such as to *acos* or *sqrt*. In Section 3 we explain Penny Pincher in detail. Our evaluation in Section 4 shows that although our method is deceptively simple, we still achieve high accuracy for a variety of datasets with a limited number of templates and superior accuracy under an imposed time constraint. Finally we conclude the paper in Sections 5 and 6.

## 2 Related Work

Since Rubine's [13] seminal work on gesture recognition, numerous techniques have been developed for both gesture and symbol recognition. One popular branch of research focuses on easy to understand and easy to implement recognizers. The $-family of recognizers started with Wobbrock et al. [17] who designed the $1 recognizer. This simple and effective unistroke recognizer works by resampling strokes to a predetermined number of points, and then these strokes are translated and scaled to the unit square centered at zero. Euclidean distance between points is then used as a similarity metric to compare candidate strokes with template gesture strokes. The template with the least distance is assumed to be the most likely gesture. To ensure the best match possible, $1 also uses a golden section search (GSS) [11] to rotate candidate strokes into alignment with the template under consideration. Anthony and Wobbrock [2] extended the $1 unistroke recognizer into the $N multistroke recognizer. This version is able to handle multistroke gestures by combining all strokes into a single stroke so that $1 techniques can be leveraged. Given an example gesture, during training when templates are constructed, all permutations of strokes and stroke directions are used to generate all possible ways in which the gesture can be written.

While the previous approaches work with arcs, Vatavu et al. [16] considers input as point clouds. The $P recognizer still resamples, scales, and translates strokes and uses Euclidean distances as a similarity metric. However, the points are not treated as a time series; instead the authors employ a greedy algorithm that tests different permutations of points to find the overall minimum distance between candidate and template point clouds. Herold and Stahovich [6], on the other hand, continue to work with strokes as time series, but their 1¢ recognizer generates a rotation invariant representation that is a time series vector of one-dimensional points—normalized distances measured from the stroke's centroid. That is, strokes are resampled as usual, but then each two-dimensional point is converted into a one-dimensional scalar, the distance of the point from the centroid normalized by the standard deviation of all distances. Candidate and template strokes are then compared using the one-dimensional Euclidean distance.

One major limitation of $1, $N, and $P is that they are generally slow. The former two recognizers are slow due to their use of the GSS. Li [9] was able to overcome this limitation. He discovered a closed form equation to find the optimal rotation of a candidate stroke to match the template. Since Li's Protractor recognizer represents each stroke as a vector and works with the angle between the two vectors (between the candidate and template vectors), it is not necessary to scale candidate strokes when matching. However, resampling and translation to the centroid are still required. Anthony and Wobbrock [3] adapted Protractor's technique to create a fast multistroke recognizer, $N-Protractor.

To further speed up template matching, Reaver et al. [12] proposed Quick$ to use agglomerative hierarchical clustering with a branch and bound search to cull branches that cannot contain a better match than what has already been found. Zhang et al. [18] use nonlinear embedding [7, 8] as a filter to quickly check the lower boundary of the Euclidean distance between a candidate and template using only stroke means and variances based on a low dimensional representation. Vatavu [14] developed 1F to sort all templates based on a scalar value (such as total curvature). Then the probability mass functions (*pmf*) are learned for each template. During the matching process, the best match is identified based on the 1F value, and then the *pmf* of the associated template is used to determine how far left and right of the current template to search for the best match. Except for nonlinear embedding, our Penny Pincher recognizer is orthogonal to these methods. However, such methods are not within the spirit of the $-family because of their additional complexity in code and data structures.

Compared to other $-family recognizers, Penny Pincher is the easiest yet to understand and simplest to implement. As mentioned previously, we do not rotate, scale, or translate candidate gestures, whereas every other recognizer does at least one of these. Also, most recognizers use Euclidean distance and have to make calls to geometric functions at some point in the template matching process. We compare between-point vectors instead, so that only simple dot product calculations are required (after the gesture has been resampled). As we show later, these simple modifications improve computational performance without sacrificing accuracy.

## 3 Penny Pincher

Loosely following [14], we define a gesture as an ordered series of points as follows:

$$g = \left\{ g_i = (x_i, y_i) \in \mathbb{R}^2 \right\}, \tag{1}$$

where $i$ indexes the gesture's points in time relative order. Gestures are resampled by length into $n$ points so that the distance between each point is equal. A gesture is then converted into a series of between-point vectors:

$$v = \left\{ v_i = g_{i+1} - g_i \right\}, \tag{2}$$

for $i < n$. Given two gestures $v$ and $w$, their similarity is taken to be the sum of the angles between corresponding between-point vectors:

$$D(v, w) = \sum_{i=1}^{n-1} \frac{v_i \cdot w_i}{\|v_i\| \, \|w_i\|}. \tag{3}$$

A perfect score is therefore $n-1$ because the normalized dot product is 1 for identical vectors. Now, let $m$ be the number of distinct gestures and $\mathscr{T}$ the set of templates created by training the recognizer:

$$\mathscr{T} = \{t_i = (s_i, l_i) \mid l_i \in \{1 \ldots m\}\}, \tag{4}$$

where $t_i$ is the enumerate template in set $\mathscr{T}$, $s_i$ is the gesture sample's between-point vector, and $l_i$ is the gesture's class label. A candidate gesture $c$ takes the classification of the template $T$ that is most similar:

$$T = \underset{t_i \in \mathscr{T}}{\arg\max} \ D(t_i, c), \tag{5}$$

This already simple calculation can be simplified further. During training, let the components of the between-point vectors be normalized so that each $\|s_i\| = 1$, thus eliminating one normalization factor in Equation 3. Next, as a simplifying assumption, say that the length of each between-point vector component is equal (each $\|w_i\| = \|w_{i+1}\|$) because gestures are resampled into $n-1$ equidistance arc lengths; this assumption is evaluated in Section 4.2. This allows us to treat $\|c_i\|^{-1}$ as a constant to be factored out of the summation of Equation 3. Further, note that this factor is the same for all evaluated templates in Equation 5—it scales each dot product result identically. Therefore the scaling factor can be eliminated entirely without affecting the final result, thus simplifying our template matching equation to:

$$T = \underset{t_i \in \mathscr{T}}{\arg\max} \sum_{i=1}^{n-1} v_i \cdot w_i. \tag{6}$$

There are a few distinct advantages to this formulation. Notice first that there are no rotation, scale, or translation operations involved. This means less overhead in preparing templates and matching. Also, template matching is reduced to merely addition and multiplication with no expensive calls to geometric library routines. This reduction in overhead compared to other recognizers means that more templates can be evaluated in the same amount of time, see Section 4. The simplicity of our approach also means Penny Pincher is the easiest to understand and implement in the \$-family.

It should be noted that Penny Pincher is not specifically a multistroke gesture recognizer. However, we leverage the same technique used by \$N [2], which is to concatenate sequential strokes together into a single unistroke gesture. We find that with a sufficient number of training samples, this achieves the same effect as \$N's scheme to internally generate all possible permutations of a gesture from a single sample.

## 3.1 Complexity

In this section we discuss the theoretical minimum asymptotic boundary of the \$-family of recognizers. Without the use of filtering or other culling techniques we make the following assumptions:

1. Candidate gestures are always resampled.
2. The resampled candidate gesture is always compared against every template in $\mathscr{T}$.
3. Each point in the resampled candidate gesture is evaluated against at least one corresponding template point.

Under these assumptions, let $r = |c|$ be the length of the candidate gesture before resampling, representing the number of raw data points. As before, $n$ is the number of resampled data points. The cost of resampling $c$ is the cost of first calculating the stroke length and then computing the resample points. This requires touching every raw data point twice, $\Omega(2r)$, and every resample point once, $\Omega(n)$. The latter is because each resample point is part of the path and must be considered as the resampling procedure continues. Supposing that the recognizer can match the resampled points directly (e.g., without further manipulation), then no additional processing is necessary. Template matching is subsequently carried out for each $t \in \mathscr{T}$, which is $\Omega(n|\mathscr{T}|)$. Therefore, at best, a template based recognizer as described is bounded by:

$$\Omega(2r + n + n|\mathscr{T}|). \tag{7}$$

To the best of our knowledge, Penny Pincher is the first \$-family recognizer to achieve this lower bound. Note that the complexity of several other recognizers are available in [16], however, these do not consider the cost of resampling. Of course $\Omega$ notation hides certain details that are important in practice. We have to consider that 1¢, for example, represents each point in its template as a one-dimensional point where the other methods use two-dimensional points. This means that 1¢ can compare templates faster despite having a non optimal resampling strategy as is shown in the evaluation Section 4.3. Penny Pincher, on the other hand, relies on straightforward dot product calculations without having to utilize external or built-in math libraries.

## 4 EVALUATION

We evaluate Penny Pincher using three different tests. All tests were performed on a MacBook Pro with a 2.4 GHz Intel Core i7 processor, 8 GB of 1333 MHz DDR3 memory, and an 760 GB mechanical SATA disk. In our first test we check the validity of the assumption that all between-point vector lengths of a single gesture are of equal length. Next we evaluate the accuracy of our method compared to other \$-family recognizers when the number of templates is controlled (the standard method of evaluation). Finally we investigate the accuracy of the fastest recognizers to see how well they perform under varying time constraints. However, we first describe the various datasets used in our tests in the follow subsections.

## 4.1 Datasets

This subsection gives a brief description of each dataset used in our evaluation. For additional information beyond what is presented here, we refer the reader to the associated cited work.

**\$1-GDS**. The \$1 recognizer gesture dataset [17] is a unistroke gesture dataset comprising 16 gestures collected from 10 subjects at 3 speeds (slow, medium, and fast) on an HP iPAQ h4334 Pocket PC. Because each subject supplied 10 samples of each gesture at each speed, there are 4800 samples in aggregate.

**SIGN**. The Synchromedia-Imadoc Gesture New On-Line Database [1] is a unistroke gesture dataset comprising 17 classes collected from 20 writers on tablet PCs and whiteboards. Each writer supplied 25 samples over four sessions so that SIGN contains 8500 samples in aggregate.

**EDS 1** and **EDS 2**. Two unistroke gesture datasets were collected by Vatavu et al. [15] to study gesture execution difficulty, referred to as Execution Difficulty Set #1 (EDS 1) and Execution Difficulty Set #2 (EDS 2). The first dataset contains 5040 samples in aggregate comprising 18 gestures collected from 14 participants providing 20 samples each. The latter dataset contains 4400 samples in aggregate comprising 20 gestures from 11 participants providing 20 samples each. All samples were collected on a Wacom DTU-710 Interactive Display.

**MMG**. The Mixed Multistroke Gestures dataset [3] comprises 16 gestures having between one and three strokes. Samples were collected from 20 participants using either a stylus or their finger on a tablet PC at three different speeds (slow, medium, and fast). In aggregate there are 9600 samples.

**UJI**. The UJIpenchars2 Database [10] is a multistroke gesture dataset of lowercase and uppercase letters, digits, characters with diacritics, and punctuation marks containing an aggregate of 11640 samples over 97 gesture classes. Data was collected from 60 writers each providing 2 samples of each class.

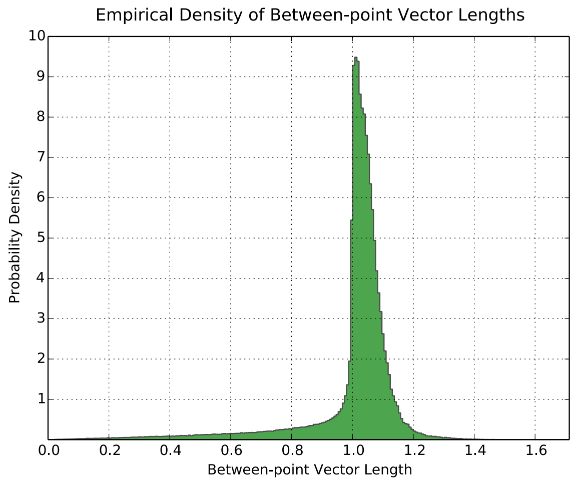

**Figure 2:** The empirical probability density of the distribution of between-point vector length. For each resampled gesture in the $1-GDS, SIGN, EDS 1, and EDS 2 datasets, we normalize the between-point vectors to the average length of the vectors within the gesture. All results are combined to create this overall distribution of relative lengths. It can be seen that the majority of the density is centered around 1.0 which indicates that the majority of between-point vectors are of equal length within each sample.

### 4.2 Verification of Between-Point Vector Distribution

The simplifying assumption used in Equation 6 supposes that each between-point vector is of equal length. In reality, this is inaccurate because indeed, vectors taken from collinear points are longer than those taken from curves, which in turn are generally longer than those taken from sharp cusps. However, for the purpose of an approximation, we find this assumption to be sufficient. To verify this we examine the empirical probability density distribution generated with the relative between-point vector lengths of each sample in six datasets ($1-GDS, SIGN, EDS 1, EDS 2, MMG, and UJI). For each individual sample, the mean between-point vector length is computed and all vectors within the same sample are normalized by the mean. Then the set of all normalized vectors from all sets are combined to build the empirical probability density distribution shown in Figure 2. If all between-point vectors were truly of equal length then the full density would be located at one. We see, though, that the majority of the density is near one with the distribution being left skewed. Specifically, 85% of the distribution is contained between 0.9 and 1.15 and is left skewed $-2.85$. The individual distributions of the six unique datasets are all similar to the overall distribution. They are left skewed in the same manner (-3.08, -3.06, -2.44, -2.56, -2.5, and -2.59 respectively) and contain a similar portion of the density within the same range (87.58%, 87.43%, 79.96%, 85.83%, 78.81% and 81.06% respectively). Further, note that the observed left skewness is expected. When most vectors are of similar length but there are a small number of short vectors on sharp cusps for example, then the mean is pulled down, shifting the densest portion of the distribution right of the mean. Nevertheless, given that the majority of the density is located near one, we find our previous assumption that between-point vector lengths are approximately equal to be accurate enough for our use.

### 4.3 Standard Test

Using all six datasets we compared the accuracy of Penny Pincher against its predecessors $1, Protractor, $N, $N-Protractor, and 1¢ using what we call the standard test. The resampling rate of each was set according to the author's recommendations: 1¢, Protractor, and Penny Pincher resample to 16 points whereas $N, $N-Protractor, and $1 resample to 96 points. Where applicable, we enabled bounded rotation invariance due to an observed slightly higher recognition performance. Note that we also considered $P. However, we found that the recognizer was very slow in practice, and based on results reported in [16], this recognizer performs similarly to $1 for unistroke gestures. Also because we are interested in recognizers that can potentially process numerous templates in a short amount of time, $P was not an option.

In this part of the evaluation, the template count $T$ varies from 1 to 10 for each gesture and we consider only user independent scenarios. For each template count and for each recognizer we ran 1000 tests. That is, in each iteration we randomly select $T$ templates per gesture from all of the available samples, and then we randomly select one remaining sample from each gesture to be the candidate gesture. Therefore each test contains $g$ recognition tasks, where $g$ is the number of gestures in the dataset. Throughout the following, we report on the recognition error of the various recognizers (which is equivalent to one minus the accuracy). All of the standard test results can be found in Figure 3.

Penny Pincher performs strongly on the $1-GDS dataset. With only one template loaded, the error rate is 8.58% which drops to 1.3% by ten templates. $1, on the other hand, also starts at 8.58% but only achieves a 1.9% error by the end. Protractor makes the most dramatic improvement in reduction of error by swinging from 12.16% to 1.55%, beating out $1 starting at 6 templates. For the EDS 1 dataset, Penny Pincher is mostly a middle of the road yet still accurate performer, achieving a 3.3% error with one template, and dropping to below 0.7% with three templates, but reaching the best result of all recognizers at 0.11% error with ten templates loaded. With EDS 2, our recognizer starts at 2.19% and drops to .70% by the second template. For the remaining template counts, Penny Pincher maintains the highest accuracy with its best result at 9 templates (0.11% error). Both $1 and Penny Pincher perform very well compared to $N and Protractor for the SIGN dataset, although Penny Pincher has better overall performance. We start at a 13.41% error which drops over ten templates to 2.2%, whereas $N starts at 13.47% and only drops to 4.41%

MMG is a dataset designed for $N and $N-Protractor which were developed to handle multistroke gestures. So it is expected that Penny Pincher would not perform as well in this case. Our recognizer is approximately in line with Protractor, where with one template loaded, we see 38.31% error. This steadily decreases until the error rate reaches 8.0% using 10 templates. One thing to note, however, is that $N internally creates a template for every possible permutation of the strokes and their directions. So although the template count is 10, for example, in reality $N is evaluating significantly more templates. As is shown in the third part of our evaluation, Penny Pincher is also capable of achieving similar or better results when more templates are available. UJI is a large gesture dataset that has gestures with significant similarities, making it a difficult dataset for all recognizers. Our recognizer starts out at a 67.12% error rate and only achieves 41.53% error with ten templates loaded. This is approximately in line with with $1 and Protractor. $N, however, fares less well, only reaching 45.32%

In all of the six datasets, 1¢'s accuracy was well below the other recognizers. This is likely due to the fact that 1¢ is completely rotation invariant, and while this turns out to be good for computational performance, the effect is that it is not a good general purpose recognizer. Table 1 shows computational performance of the fastest recognizers. This was determined by averaging together the duration of each of the individual tests for the UJI dataset for ten templates, which was then divided by the total number of loaded templates ($T * g$). The result is the time it takes to compare one candi-

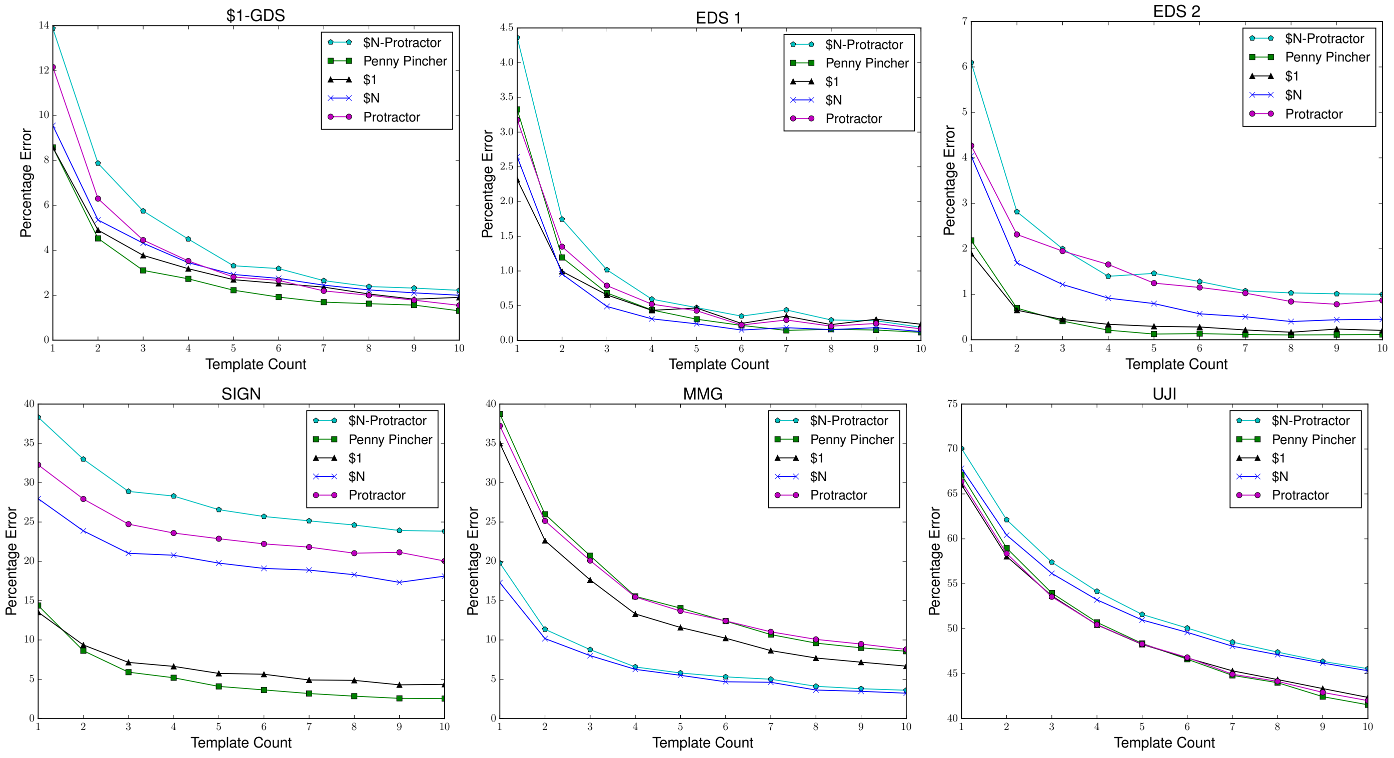

**Figure 3:** User independent error recognition test results for varying template counts. Each test, for each recognizer and template count, was performed 1000 times. 1¢ was also tested, though because it exhibited high error rates with these datasets, it was removed for readability.

| Recognizer | Avg ns / Template | ns-$\sigma$ |
|---|---|---|
| 1¢ | 25 | 4.5 |
| Penny Pincher | 33 | 6.2 |
| Protractor | 174 | 27.7 |
| $N-Protractor | 2449 | 650 |

**Table 1:** Average time taken in nanoseconds per template to perform the recognition task for the UJI dataset given 10 templates per gesture. The four fastest recognizers are shown alongside their standard deviations. Although 1¢ is the fastest recognizer, its accuracy is subpar compared to its competitors for this dataset.

date gesture to one template gesture (though note that the cost of resampling the candidate gesture is amortized over all comparisons). 1¢ is the fastest recognizer being able to compare two gestures in 25 nanoseconds. Penny Pincher is also vey fast, achieving a 33 nanosecond benchmark. Though 1¢ is faster, its accuracy prohibits its general use as we saw when working with the six test datasets. Finally, Protractor, relative to the top two recognizers, is slow in comparison, taking 174 nanoseconds to complete one check, which is 5.27 times Penny Pincher. This difference in speed is one reason why Penny Pincher is able to achieve high accuracy as reported in the next subsection.

### 4.4 Budget Test

We refer to our final test as the budget test. In this scenario we are given a time constraint and are allowed to process as many templates as possible within the given boundary. This setup is similar to the standard test except that we vary the time constraint rather than the template count directly. For each test we first determine the number of templates the recognizer can process per second. We then start with a 10$\mu$s budget and allow the recognizer to train with as many templates as it can process within the time constraint. Using this configuration, as before, we execute 1000 randomized tests. Thereafter the budget is incremented by 10$\mu$s and the test is rerun. This continues until there are not enough remaining samples to perform a full recognition test. Since the number of templates a recognizer can train with depends on the budget, it is possible to have an uneven number of templates per gesture class. If, for example, the budget allows for 2.2 templates per gesture, we ensure that at least two templates are loaded per class and then a random 20% of the gesture classes will have a third template loaded. The results of this test are shown in Figure 4. The graphs are cropped to 100$\mu$s for readability, but the tail results are reported below.

Penny Pincher achieves high accuracy for EDS 2 right at the start with a 0.13% error and is able to process approximately 7.9 templates per gesture (tpg) within 11$\mu$s. As the test continues, the dataset is exhausted at a 111.29$\mu$s budget where the error rate drops to 0.06% and 208.6 tpg are processed. 1¢ initially achieves a 3.2% error with 10.4 tpg at a 11$\mu$s budget. This improves to only 0.53%. Protractor starts with the worst accuracy, only being able to handle 1.05 tpg at 10$\mu$s. As expected, Protractor takes the longest to complete where the dataset is exhausted at 701.34$\mu$s, though the error rate drops to 0.1%. In all budgets, Penny Pincher achieves a considerably lower error rate.

Of all our tests, 1¢ achieves its best results with the $1-GDS budget test. Although the initial error is high (14% at 19.12 tpg), this drops to 0% by the end with 297.94 tpg at 90$\mu$s. Penny Pincher also reaches 0% with 219.38 tpg at 89$\mu$s. However, its initial error is much better, 0.85% with 19.12 tpg at 11$\mu$s. Starting with 2.81 tpg, Protractor only achieves a 5.3% error and at its best, cannot reach 0%. Using 297.56 tpg over 766$\mu$s, Protractor reaches .2% error.

UJI again is the most challenging dataset. Even at 65.81 tpg and

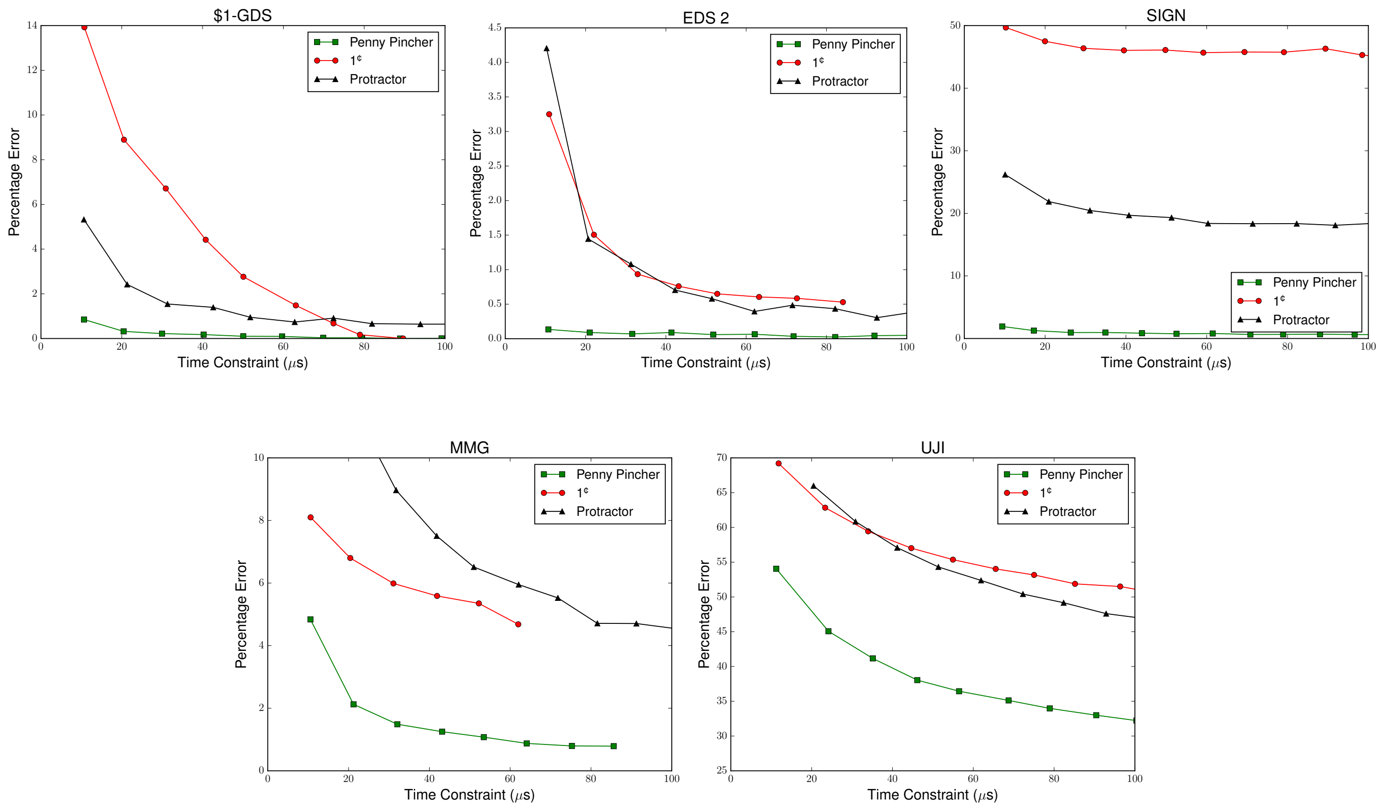

**Figure 4:** User independent error recognition test results for varying time constraints. Each test, for each recognizer and budget, was performed 1000 times. Although the graphs are cropped to 100$\mu$s for readability, the tail results are reported in Section 4.4.

| | **20$\mu$s** | | **40$\mu$s** | | **60$\mu$s** | | **80$\mu$s** | | **100$\mu$s** | |
|---|---|---|---|---|---|---|---|---|---|---|
| $1-GDS | 87% | (0.32, 2.41) | 87% | (0.18, 1.39) | 87% | (0.09, 0.74) | 95% | (0.03, 0.66) | 99% | (0.01, 0.64) |
| EDS 2 | 94% | (0.09, 1.45) | 87% | (0.09, 0.71) | 84% | (0.07, 0.40) | 94% | (0.03, 0.44) | 87% | (0.05, 0.38) |
| SIGN | 94% | (1.26, 21.88) | 95% | (0.96, 19.69) | 96% | (0.75, 18.38) | 96% | (0.65, 18.36) | 96% | (0.69, 18.40) |
| MMG | 69% | (2.13, 6.80) | 83% | (1.25, 7.51) | 85% | (0.88, 5.95) | 83% | (0.79, 4.71) | — | — |
| UJI | 26% | (45.07, 60.84) | 30% | (38.04, 54.32) | 30% | (35.12, 50.42) | 31% | (32.99, 47.60) | 31% | (31.41, 45.7) |

**Table 2:** Reduction in percentage error for 20, 40, 60, 80 and 100 microsecond budgets. Shaded cells show the percentage reduction in error achieved by Penny Pincher compared to the second most accurate recognizer at each sample location (see the individual graphs in Figure 4 to know which recognizer this is). To the right of each percentage reduction result is the exact percentage error of Penny Pincher and its competitor, in this order. Note that with UJI, the dataset is exhausted before Penny Pincher reachers 100$\mu$s, which is why there is no entry.

135$\mu$s, the error only drops to 49% from 69% for 1¢. Protractor is a little better, getting down to 30% from 66% but also requiring a 1152$\mu$s budget to do so. Penny Pincher has the best performance overall. With 3.06 tpg at 11$\mu$s, the initial error is 54%, which steadily drops to 28% by 183$\mu$s.

Initially Penny Pincher did not do as well as $N when working with the MMG multistroke dataset. However, as discussed, $N internally creates the various permutations of a gesture from a single template, which implies that given a specific template count, the recognizer is actually processing significantly more variations. In this scenario, that is no longer true. Each recognizer is trained with numerous examples as afforded by the budget so that most practical variations of the gestures are learned by the recognizer (whereas $N may generate permutations that never occur in practice). Therefore Penny Pincher working with 18.75 tpg at 11$\mu$s can achieve a 4.8% error, which declines rapidly to 1.1% (42.19 tpg) at 21$\mu$s, and then to 0.79% at 86$\mu$s (183.06 tpg). Protractor and 1¢ also see higher accuracies but do not fare as well as Penny Pincher. Respectively their best errors rates are 2.9% and 4.7% (198 tpg at 526$\mu$s and 187 tpg at 62$\mu$s).

SIGN again is where we see the most dramatic difference in performance between Penny Pincher and the other recognizers. With a budget of 9$\mu$s and 14.29 tpg, the error is already 1.9%. This continues to improve until the end where the error drops to 0.54% (with 1936 tpg over 1213 $\mu$s). 1¢ struggles initially in comparison with a 50% error that only drops to 43% near the end with 747$\mu$s. Protractor is in the middle with an initial 26% error (2.41 tpg) which improves only to 14% with a 5720$\mu$s budget.

In Table 2 we provide a summary of select results from the budget test. For each sample, we compare Penny Pincher to the second best performing recognizer in the category and report the percentage reduction in recognition error. The exact errors are shown next to each result. It can be seen that in all cases, Penny Pincher significantly outperforms the other recognizers. With four datasets, the reduction in recognition error ranges between 69% and 99% with most results being above 83%. Our worst result is observed with the UJI dataset where the recognition error is only improved by 26—31%.

## 5 Discussion and Future Work

Penny Pincher was designed to be as fast as possible and we took a number of steps to achieve this goal. First, we decided to base our similarity metric on the dot product of between-point vectors. This immediately afforded us two benefits, that candidate gestures neither need to be scaled nor do they have to be translated to a common origin for comparison. Second, we worked to eliminate the need for normalization during template matching. This is accomplished by normalizing the between-point vectors of template gestures during training, and by assuming that the distance between points of resampled strokes are equal. With this approach, we reduced the recognition task down to just addition and multiplication. As a result, Penny Pincher is the simplest and easiest to understand of all \$-family recognizers. In terms of speed, only 1¢ is faster, but as was demonstrated, it seems that 1¢ is not well suited for general purpose gesture recognition. In terms of accuracy, for three datasets we achieved the lowest user independent recognition error with accuracies between 97.5% and 99.9%. Over an additional three datasets, our recognizer still remained competitive, indicating that Penny Pincher is an excellent general purpose recognizer even without considering speed.

Computational performance, however, was our primary concern. Given a difficult time budget, we worked to develop a recognizer that could process as many templates as possible to achieve the highest recognition rate possible within the specified constraint. In the budget portion of our evaluation we demonstrated that Penny Pincher succeeded in this objective. With our test apparatus, Penny Pincher is able to process approximately 30.3 templates per microsecond, whereas Protractor (generally the second most accurate recognizer in the budget test) is limited to 5.7. Therefore, our recognizer is able to achieve a percentage error that is less than 1% for all but one of the tested datasets given as little as $60\mu$s. When considering reduction in recognition error for the various time budgets, Penny Pincher sees an 83% or higher reduction in most cases, and in several instances, the reduction is 94% or more. The UJI dataset is by far the most difficult to work with because of the similarity in gestures (e.g, upper- vs lower-case 'O') and because the number of gesture classes is large (97). However, the reduction in error still ranges between 26% and 31%. This indicates that from all of the \$-family recognizers, ours is best suited for tight time budgets, when a large number of samples are available. Also note that our budgets were selected because of the number of examples available in the datasets and because of our apparatus. In other environments, such as an implementation in an interpreted language running on a mobile device, budgets would likely have to be much higher to attain the same accuracies.

Our derivation of the recognizer assumes that the between-point vectors are of equal length. While this is not strictly true, the empirical probability distribution of relative between-point vector lengths shows that as an approximation the assumption is valid. But because the distribution is left skewed, what does this really mean for shorter vectors with respect to pattern matching? In a resampled stroke, between-point vectors on straight lines will have the longest length. Relative to shorter vectors that fall on cusps, straight lines will have a larger influence on the final estimate. Therefore, under our assumption, Penny Pincher winds up weighting straight lines with greater importance. Interestingly, in informal ad hoc testing, we saw that this had a positive impact on accuracy and as part of future work, it is worth investigating further to determine how lines, curves, and cusps impact accuracy with respect to our method.

In our evaluation we demonstrated the benefits of Penny Pincher, but it is also important to understand its various limitations. Since we compare between-point vectors, Penny Pincher is scale invariant, though unlike other recognizers it is not rotation invariant. As is turns out though, with the datasets we evaluated, this was not an issue and with more templates loaded, variations in angular displacement are well represented. Our recognizer is also not intrinsically a multistroke recognizer. Similar to \$N, we concatenate multiple strokes together to create a single unistroke gesture for template matching, but unlike \$N, we do not generate all of a sample's various permutations. Instead, training Penny Pincher with additional examples essentially has the same effect and therefore, it is not necessarily pertinent to derive the numerous permutations (some of which may never occur in practice anyways) to achieve high accuracy.

The idea behind our work is that there are potentially hundreds of templates available per gestures for the recognizer to utilize, but this poses two possible problems. First, a large amount of memory is required to store high counts of template data. Suppose that an implementation is using double-precision floating point numbers that are 8 bytes long. Each two-dimensional point then requires 16 bytes and with a sixteen point resampling count, each template requires 256 bytes of memory (not including any overhead needed for the template structure itself). Given sixteen gestures and two-hundred samples per gesture, approximately 800 KiB of memory are needed to store all of the data. For most systems, 800 KiB is negligible, but in some cases this may be impractical. The second issue is that in certain applications there are simply not that many examples available. This will occur, for instance, if a user is supplying custom gestures to a user interface. As we saw in the evaluation though, Penny Pincher provides high accuracy already with even a small number of templates. Further, over time as the system is used, additional examples can be collected and the recognizer can be continuously retrained with new data.

As part of future work, in the spirit of making things go fast, it would be interesting to look at alternatives to equidistant resampling of candidate gestures. With thousands of templates loaded, the majority of time is spent in matching, but by eliminating the last remaining geometric library calls (namely *sqrt*), there may be a small positive effect. More importantly though, not all templates give relevant additional information about the sample space. For example, if a minimum subset of the available templates are sufficient to define a portion of the sample subspace (so that if any candidate gesture falls within this subspace, it will be classified correctly), then any additional templates in this subspace are superfluous. Therefore it would be interesting to research template selection techniques that are appropriate for Penny Pincher and that are also within the theme of the \$-family.

## 6 Conclusion

We have presented Penny Pincher, a fast and accurate \$-family gesture recognizer that compares between-point vectors to estimate similarity. This recognizer is stripped down to the bare essentials so that after a gesture is resampled, only elementary arithmetic is required to achieve great performance. In a user independent evaluation utilizing six unique datasets of varying complexity, we demonstrated that Penny Pincher outperforms \$1, Protractor, \$N, \$N-Protractor, and 1¢ in three cases with just a small number of templates (achieving 97.5%, 99.8%, and 99.9% accuracy), and remains competitive in the other cases. In a budget test where each recognizer is given a limited amount of time to perform its recognition task, Penny Pincher significantly outperforms the other recognizers, achieving a reduction in recognition error of between 83% and 99% with four datasets, and a 31% reduction with the most difficult dataset.

## Acknowledgements

This work is supported in part by NSF CAREER award IIS-0845921 and NSF award CCF-1012056. Also, we would like to thank the members of ISUE lab for their support and the anonymous reviewers for their useful comments and feedback.

## References

[1] A. Almaksour, E. Anquetil, S. Quiniou, and M. Cheriet. Personalizable pen-based interface using lifelong learning. In *2010 International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 188–193, Nov 2010.

[2] L. Anthony and J. O. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010*, GI '10, pages 245–252, Toronto, Ont., Canada, Canada, 2010. Canadian Information Processing Society. ISBN 978-1-56881-712-5.

[3] L. Anthony and J. O. Wobbrock. $n-protractor: A fast and accurate multistroke recognizer. In *Proceedings of Graphics Interface 2012*, GI '12, pages 117–120, Toronto, Ont., Canada, Canada, 2012. Canadian Information Processing Society. ISBN 978-1-4503-1420-6.

[4] W. D. Gray and D. A. Boehm-Davis. Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of experimental psychology: Applied*, 6: 322–335, 2000.

[5] J. Gregory. *Game Engine Architecture*. Ak Peters Series. Taylor & Francis, 2009. ISBN 9781568814131. URL `http://books.google.com/books?id=LJ20tsePKk4C`.

[6] J. Herold and T. F. Stahovich. The 1&cent; recognizer: A fast, accurate, and easy-to-implement handwritten gesture recognition technique. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, SBIM '12, pages 39–46, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association. ISBN 978-3-905674-42-2.

[7] Y. Hwang and H. kap Ahn. Convergent bounds on the euclidean distance. In J. Shawe-taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 388–396. 2011.

[8] Y. Hwang, B. Han, and H.-K. Ahn. A fast nearest neighbor search algorithm by nonlinear embedding. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3053–3060, June 2012.

[9] Y. Li. Protractor: A fast and accurate gesture recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2169–2172, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9.

[10] D. Llorens, F. Prat, A. Marzal, J. M. Vilar, M. J. Castro, J.-C. Amengual, S. Barrachina, A. Castellanos, S. E. Boquera, J. Gómez, et al. The ujipenchars database: a pen-based database of isolated handwritten characters. In B. M. J. M. J. O. S. P. D. T. Nicoletta Calzolari (Conference Chair), Khalid Choukri, editor, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may 2008. European Language Resources Association (ELRA). ISBN 2-9517408-4-0.

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[12] J. Reaver, T. F. Stahovich, and J. Herold. How to make a quick$: Using hierarchical clustering to improve the efficiency of the dollar recognizer. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 103–108, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0906-6.

[13] D. Rubine. Specifying gestures by example. *SIGGRAPH Computer Graphics*, 25(4):329–337, July 1991. ISSN 0097-8930.

[14] R.-D. Vatavu. 1f: One accessory feature design for gesture recognizers. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces*, IUI '12, pages 297–300, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1048-2.

[15] R.-D. Vatavu, D. Vogel, G. Casiez, and L. Grisoni. Estimating the perceived difficulty of pen gestures. In *Proceedings of the 13th IFIP TC 13 International Conference on Human-computer Interaction - Volume Part II*, INTERACT'11, pages 89–106, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-23770-6.

[16] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock. Gestures as point clouds: A $p recognizer for user interface prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction*, ICMI '12, pages 273–280, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1467-1.

[17] J. O. Wobbrock, A. D. Wilson, and Y. Li. 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, pages 159–168, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-679-0.

[18] Y. Zhang, W. Deng, H. Song, and L. Wu. A fast pen gesture matching method based on nonlinear embedding. In T. Tan, Q. Ruan, X. Chen, H. Ma, and L. Wang, editors, *Advances in Image and Graphics Technologies*, volume 363 of *Communications in Computer and Information Science*, pages 223–231. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-37148-6.

## A Penny Pincher Psuedocode

Note that Resample-Between-Points and Path-Length are adapted from [17]. Also, addition, multiplication, and division on points are element-wise (i.e., $(x,y)$) operations.

```
Resample-Between-Points (Points points)
1:  $I \leftarrow$ Path-Length$(A)$ / $(n-1)$
2:  $D \leftarrow 0$, $v = \{\}$
3:  $prev \leftarrow points_0$
4:  foreach $p_i$ in $points$ such that $i \geq 1$ do
5:      $d \leftarrow$ Distance$(p_i, p_{i-1})$
6:      if $D + d \geq I$ then
7:          $q \leftarrow p_{i-1} + (p_i - p_{i-1}) * (I - D)/d$
8:          $r \leftarrow q - prev$                          ▷ Equation 2
9:          $r \leftarrow r$ / Distance$((0,0), r)$           ▷ Only if template
10:         $D \leftarrow 0$, $prev \leftarrow q$
11:         Append$(v, r)$
12:         Insert$(points, i, q)$
13:     else $D \leftarrow D + d$
14: return $v$
```

```
Path-Length (Points points)
1: $d \leftarrow 0$
2: for $i \leftarrow 1$ to $|points| - 1$ do
3:     $d \leftarrow d +$ Distance$(points_{i-1}, points_i)$
4: return $d$
```

```
Distance (Point a, Point b)
1: return $\sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$
```

```
Recognize (Points g, Templates $\mathcal{T}$)
1:  $c \leftarrow$ Resample-Between-Points$(g)$
2:  $similarity \leftarrow -\infty$
3:  foreach $t$ in $\mathcal{T}$ do
4:      $d \leftarrow 0$
5:      for $i \leftarrow 0$ to $n - 2$ do
6:          $d \leftarrow d + t_{i_x} c_{i_x} + t_{i_y} c_{i_y}$      ▷ Equation 6
7:      if $d > similarity$ then
8:          $similarity \leftarrow d$
9:          $T \leftarrow t$
10: return $\langle T, similarity \rangle$
```