# The Voight-Kampff Machine for Automatic Custom Gesture Rejection Threshold Selection

Eugene M. Taranta II
University of Central Florida
Orlando, FL, USA
etaranta@gmail.com

Mykola Maslych
University of Central Florida
Orlando, FL, USA
maslychm@knights.ucf.edu

Ryan Ghamandi
University of Central Florida
Orlando, Florida, USA
ryanghamandi1@gmail.com

Joseph J. LaViola Jr.
University of Central Florida
Orlando, Florida, USA
jjl@cs.ucf.edu

## ABSTRACT

Gesture recognition systems using nearest neighbor pattern matching are able to distinguish gesture from non-gesture actions by rejecting input whose recognition scores are poor. However, in the context of gesture customization, where training data is sparse, learning a tight rejection threshold that maximizes accuracy in the presence of continuous high activity (HA) data is a challenging problem. To this end, we present the Voight-Kampff Machine (VKM), a novel approach for rejection threshold selection. VKM uses new synthetic data techniques to select an initial threshold that the system thereafter adjusts based on the training set size and expected gesture production variability. We pair VKM with a state-of-the-art custom gesture segmenter and recognizer to evaluate our system across several HA datasets, where gestures are interleaved with non-gesture actions. Compared to alternative rejection threshold selection techniques, we show that our approach is the only one that consistently achieves high performance.

## CCS CONCEPTS

• **Human-centered computing** → **Gestural input**; **User interface programming**.

## KEYWORDS

gesture, customization, recognition, rejection

**ACM Reference Format:**
Eugene M. Taranta II, Mykola Maslych, Ryan Ghamandi, and Joseph J. LaViola Jr.. 2022. The Voight-Kampff Machine for Automatic Custom Gesture Rejection Threshold Selection. In *CHI Conference on Human Factors in Computing Systems (CHI '22), April 29-May 5, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3491102.3502000

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*CHI '22, April 29-May 5, 2022, New Orleans, LA, USA*
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9157-3/22/04…$15.00
https://doi.org/10.1145/3491102.3502000

## 1 INTRODUCTION

Gesture interactions have become an integral part of user interface (UI) design, and, as such, researchers have put considerable effort into gesture analysis, design, synthesis, and recognition. One particularly useful branch of research focuses on customization, where end users, including designers and developers, are able to specify gestures by example [39]. In this way, one may train their system to activate software functions using motion patterns that are personal and memorable [34]. To accommodate both customization and UI research, the community has developed a slew of light weight recognizers [2, 13, 23, 26, 44, 46, 48, 52, 53, 55, 57] referred to as the $-family and extended $-family. These recognizers collectively emphasize simplicity and straightforwardness in a way that does not sacrifice performance but promotes and facilities adaptability. For these reasons, $-family recognizers have become exceedingly popular[1]. In another line of research, given that recognizer performance generally improves with the training set size, researchers have explored data augmentation through synthesis to overcome the minimal data problem associated with customization. Techniques using the kinematic theory of rapid human movement [25] and gesture path stochastic resampling [42] have been successfully applied to this problem. The community has also investigated recognition acceleration techniques [15, 16, 38, 50], data collection protocols [40], and, recently, continuous custom gesture recognition. Machete [43], for example, is a device agnostic segmenter that is able to identify candidate gestures in a continuous stream of input. Despite these advances, an area of inquiry that remains relatively unexplored is that of rejecting malformed gestures and non-gesture input patterns, which is the main issue we address in this work.

The recognition techniques described above primarily use nearest neighbor pattern matching, which is to say that a recognizer measures the (dis)similarity of a query sample against all template samples previously demonstrated by a user. The query-template pair that yields the best score is then said to belong to the same gesture class, and the query is therefore assigned the template's gesture class label. One problem with this approach is that not all queries are gestures. This is especially apparent with continuous gesture recognition where the system must continuously decide with each new input device sample whether or not the user has

[1] See, for example, https://depts.washington.edu/acelab/proj/dollar/impact.html

gesticulated. In such systems, one may accept or reject a candidate gesture depending on its recognition score. Selecting a suitable rejection threshold, however, is problematic for several reasons.

First, it is difficult to estimate the within and between class score distributions with the amount of training data provided in customization context. Progress has been made by using synthetic data to estimate score distributions [44], though, as we later discuss, current techniques do not yield tight thresholds. In situations dealing with high activity data, where gestural interactions are interleaved with non-gesture actions, loose boundaries can lead to high false positive rates and degrade user experience [21]. Second, the optimal threshold depends, in part, on the training set size. With an infinite number of training samples, one could use an arbitrarily tight threshold, whereas, with only a single training sample, one must allow samples further away from the template to be matched. Third, Taranta *et al.* [40] recently showed that gesture production variability is application-dependent, even for the same gesture set using the same input device. As such, a threshold that is appropriate in one scenario may yield too many false negatives in another. Fourth, a threshold selection technique that works well for 2D data may not work well for 3D data. Given that 3D interactions have become increasingly mainstream through advancements in AR and VR, for example, it has become ever more important that gesture customization techniques are cross-platform compatible. The Voight-Kampff Machine (VKM) is a new rejection threshold selection technique that begins to address these issues. VKM uses synthetic data to generate score distributions from which it selects an initial rejection threshold. VKM then uses simulation based on the training set size and expected gesture production variability to adjust the threshold. Further, in this work we focus specifically on rejection threshold selection for continuous high-activity data where gesture candidates are evaluated using nearest neighbor pattern matching.

As such, the main contribution of this paper is a new device agnostic rejection threshold selection technique for continuous data that includes the following novel features:

- A new negative synthetic data generation method called Mincer for generating samples closer to the gesture boundary than prior techniques,
- A method for selecting a rejection threshold based on application-dependent gesture production variability, and
- A threshold adjustment strategy based on the training set size.

We first evaluate VKM by employing a state-of-the-art device agnostic segmenter (Machete) [43] and recognizer (Jackknife) [44] over high-activity data. The system is trained with custom gestures and then put to the task of recognizing a series of gestures embedded within a stream of constant non-gesture motion. We compare our results against an oracle rejection-threshold selection system and several alternative rejection threshold selection techniques. Our findings reveal that VKM is the only technique that achieves near optimal performance across all four input device types tested. In a second evaluation, we further compare our system against other state-other-art continuous gesture recognition approaches and show that VKM (with Machete and Jackknife) similarly outperforms its competition, thereby demonstrating that VKM is a powerful threshold selection solution. Source code, pseudocode, and additional information can be found at https://www.eecs.ucf.edu/isuelab/research/vkm/.

## 2 RELATED WORK

Our approach to rejection threshold selection is informed by several related areas of investigation, each of which we discuss throughout this section.

### 2.1 Gesture Customization

An early example of custom gesture recognition in HCI is Rubine's use of linear discriminate analysis over trivial stroke features, such as the bounding box diagonal length, stroke duration, and total angle traversed, among others [39]. Although the method was popular, it still required large quantities of data to perform well and was inaccessible to non-expert user interface designers who aspired to integrate gesture recognition into their prototype software. Both problems were addressed when Wobbrock *et al.* [57] introduced their $1 recognizer, a method that employed straightforward normalization techniques with nearest neighbor pattern matching. The success of $1 launched a branch of research into the suite of now widely used $-family and $-family like recognizers. A few examples include $N [1] for multi-stroke gestures; $P [53] and $P+ [52] for articulation invariant recognition; protractor [26], Penny Pincher [46], and $Q [55] for speed; and $1^{¢}$ [13], $3 [22], and Jackknife [44] for 3D gesture recognition. The philosophy of this work champions straightforwardness and simplicity so that such recognizers are accessible to non-experts and can easily be ported to new platforms. Although, we do not claim VKM is $-family principled, the techniques we develop herein are derived from the $-family body of work, and, in our view, complements it.

### 2.2 Gesture Production Variability

Recent research has shown a number of factors can influence gesture production variability. For instance, gestures produced within a game environment tend to be more variable than those sampled using standard data collection protocols where a user gesticulates at their convenience [8, 40, 45]. Similarly, styling words such as "perform gesture faster" can influence variability [20] as does mood [27, 30], gesture familiarity [5], and gesticulation speed [54]. It has been shown that gesture production can even vary from day to day [28]. As a result, training data variability will unlikely resemble true gesture production variability unless practitioners put effort into developing ecologically valid data collection methods. VKM takes this phenomenon into consideration when selecting a rejection threshold by using Monte Carlo simulation to inflate the rejection threshold based on expected gesture production variability differences.

### 2.3 Synthetic Data Generation

As noted, gesture recognizer accuracy generally improves with the training set size. To overcome data scarcity problems associated with gesture customization, synthetic data generation (SDG)

techniques have been proposed to increase the training set size. Various forms of SDG have been developed and implemented to achieve this, including geometric transformations [10, 47], kernel filters [18], and random erasing [58]. Even in cases of extreme data scarcity we have services like Gestures à Go Go (G3) [25], a web service practitioners may use to generate synthetic samples from real gestures. G3 uses the kinematic theory of rapid movements [37] to learn a gesture's sigma lognormal (SLM) model parameters [33], which G3 then varies to produce new samples. With this approach, one can accurately reproduce several population feature distributions from only a single example. A carefully selected SLM parameter perturbation strategy has also been used to recreate the variability of low vision users [52]. Two issues with SLM are that the technique is fairly complex and parameter extraction can have high latency. Gesture path stochastic resampling (GPSR) [42] is an alternative technique that is both $-family principled and performant. GPSR synthesizes new samples by randomly resampling points along a given trajectory, randomly removing a subset of the points, and then normalizing the Euclidean distance between each remaining pair of consecutive points. It is also interesting to note that outside of gesture customization, [11] similarly saw success with time series synthesis using random point removals and renormalization over their training data. Though SLM and GPSR have been shown to improve recognizer accuracy and fool human perception [24], we use GPSR in this work to generate positive synthetic samples since GPSR also works with 3D data [44]. However, one difference in our use of GPSR is that we define a new resampling rate equation to replicate score (rather than feature) distributions.

## 2.4 Rejection Threshold Selection

When the training dataset is sufficiently dense, there are a number ways in which one can learn a rejection threshold. Liu and Chua [29] identify three common approaches for rejecting negative samples: use a set of explicitly defined unwanted patterns to build a set of garbage models, use the distribution of scores between classes to learn a cut off, or form a universal background model (UBM) by generating mixture models from positive samples. Depending on which machine learning method is used, another option is to perform a grid search [41] or empirically tune the system through trial and error. Of course these options are difficult to apply with sparse data. For this reason, recent trends have looked to synthetic data to increase the training set size [6]. One such approach is Jackknife [44] that synthesizes gesture and non-gestures patterns from which a rejection threshold is learned. VKM takes a similar approach, though also takes factors that we previously considered into account.

Because false negatives are often more desirable than false positives [21], certain techniques dynamically adjust the rejection threshold in response to given input patterns. For example, Kang *et al.* [18] temporarily lower their rejection threshold when a pattern was rejected but is near a gesture boundary. They assume that a user will immediately retry the gesture if intended. In the presence of noise or unreliable segmentation, one may require that a gesture is detected over multiple frames before confirming recognition [4, 17, 35]. Such techniques are complementary to our work and to use VKM does not prohibit one from including these features into their system.

## 2.5 User Experience

User experience is central to gesture recognition as it determines whether a user will continue using gestures to interact with the system, or if they will revert to an input method that is more familiar and reliable. To this end, Freeman *et al.* [19] identify three considerations for building systems with gesture support: response times, algorithmic reliability, and economic constraints. In modern systems, latency and accuracy are usually the two main parameters one must balance, and, often times, one will sacrifice accuracy in favor of improved latency. For this reason, researchers have put considerable effort into accelerating recognition by employing hierarchical methods [38], data reduction [36, 49, 51], pruning [50], lower bounding [44, 55] and more efficient measurement schemes [46, 55]. However, less attention has been given to rejection, which can also be used to improve accuracy and user experience.

Katsuragawa *et al.* [21] note that false positives gestures can result in system state changes that are difficult or too time consuming to recover from. They argue that eliminating false positives is more important than eliminating false negatives. In the context of continuous custom gesture recognition over high-activity data, it is therefore critical that patterns near gesture boundaries are not falsely recognized as belonging to the same gesture class. To accomplish this, VKM introduces a new negative synthetic data generation method that produces motion patterns similar to but not precisely from the same class as a given gesture. This helps VKM select tight rejection thresholds that maximize accuracy, and, consequently, improve user experience.

# 3 THE VOIGHT-KAMPFF MACHINE

The Voight-Kampff Machine shown in Figure 1 is inspired by the rejection threshold selection system used by Jackknife (JK) [44], whereby VKM uses synthetic data generation to synthesize positive and negative score distributions from which VKM estimates a rejection threshold. In more detail, to select an initial threshold, VKM uses a variant of gesture path stochastic resampling (GPSR) to generate a common within class score distribution. VKM similarly uses a new synthetic data generation technique called Mincer to generate a common non-gesture score distribution. We refer to these distributions as the positive and negative score distributions, respectively. Using these distributions, VKM thereafter selects as the rejection threshold the score which best separates the positive and negative distributions. Two adjustments are then made to this threshold. First, we reduce the threshold based on the training set size; since more of the gesture space is covered by the training data, this reduction helps to reduce false positive errors. Second, because it has been shown that gesture variability is application-dependent [8, 40, 45], we inflate the threshold by an application-dependent gesture production variability value.

Although VKM's design is informed by Jackknife, there are several key differences. Both systems use GPSR to generate synthetic data; however, the JK authors used expert knowledge to select GPSR parameters. In our work, we use optimization to learn optimal GPSR parameters for several input device types. Another

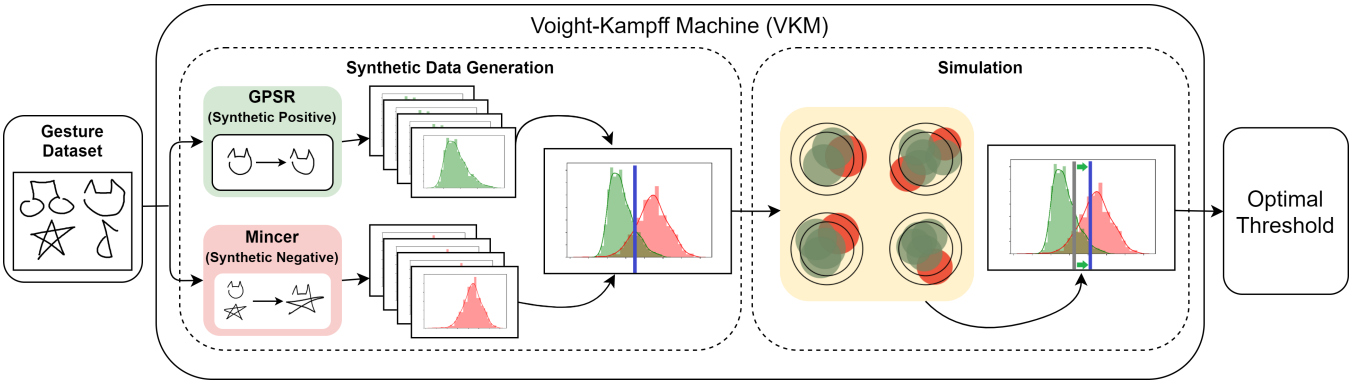


**Figure 1: Architecture diagram for the Voight-Kampff Machine. From left to right, VKM accepts a set of training samples. From each training sample, VKM generates a positive and negative synthetic data distributions using GPSR and Mincer, respectively. VKM then uses these distributions to estimate a rejection threshold that maximizes overall accuracy based on the $F_1$-score. The threshold is then adjusted using simulation to take into account application-dependent gesture production variability and the template count.**

subtle difference is that we optimize the synthetic *score* distribution rather than *feature* distributions as was done with the original GPSR work. Both systems also use synthetic data generation to synthesize negative samples. JK uses a splicing technique that generates a reasonable distribution appropriate for low-activity data under window-based segmentation, but, as we show in our evaluation, this approach is inappropriate for high-activity data. VKM, on the other hand, introduces Mincer, which is able to generate negative samples near a given gesture class boundary. This approach leads to tighter rejection thresholds. Finally, unlike prior work, VKM adjusts the threshold based on training set size and expected gesture production variability. These differences yield high quality rejection thresholds. In the remainder of this section, we describe VKM in greater detail.

## 3.1 Accuracy Measure

To select a rejection threshold, we estimate $F_\beta$-scores over distributions of synthetic data. $F_\beta$-score is a well known measure commonly used throughout the machine learning community that combines precision and recall, both of which are important in gesture recognition. $F_\beta$-scores also have the desirable property that they do not include true negative results in their calculation. This property is important because continuous input comprises mostly non-gesture data, and optimizing a system for high true negative detection may result in poor threshold selection. Formally, the $F_\beta-$score is defined as follows:

$$F_\beta = \left(1+\beta^2\right) \cdot \frac{\text{precision} \cdot \text{recall}}{\left(\beta^2 \cdot \text{precision}\right) + \text{recall}} \tag{1}$$

$$= \frac{\left(\beta^2+1\right) tp}{\left(\beta^2+1\right) tp + \beta^2 fn + fp}, \tag{2}$$

where precision is the fraction of true positives ($tp$) over all true and false positive ($fp$) samples, and recall is the fraction of true positives over all true and false negative ($fn$) samples:

$$\text{precision} = \frac{tp}{tp+fp}, \tag{3}$$

and

$$\text{recall} = \frac{tp}{tp+fn}, \tag{4}$$

where $\beta$ controls the relative weight between precision and recall. When $\beta = 1$, we have the traditional $F_1$-score that uniformly balances both measures. In our framework, false positives and true negatives are those negative samples that fall respectively left and right of a given threshold. Similarly, true positives and false negatives are those positive samples that fall left and right of the same threshold. We next describe our approach to positive synthetic data generating using GPSR, after which we introduce Mincer, our new negative synthetic data distribution method.

## 3.2 Positive Synthetic Data Generation

VKM uses GPSR [42] to generate synthetic positive score distributions. In other words, we synthesize gesture samples from training data and measure their dissimilarities from the seed sample, which results in a distribution of within class scores. Note that because we are concerned with customization, we only require that GPSR can sufficiently reproduce within class variance from a single gesture sample. Since we combine per-class distributions into a common distribution, we do not require that individual class results are realistic or accurate, only that the final distribution represents the combination of all within class distributions. We next review GPSR and thereafter discuss how we extend GPSR to support device-agnostic positive sample generation.

*3.2.1 Gesture Path Stochastic Resampling.* In short review of GPSR, one first spatially resamples a given trajectory so that the distance between selected points along a gesture path is random, rather than uniform. One then rescales the distance between points to uniform length. This process fundamentally alters the trajectory's shape, but because the transformation preserves low frequency

information, its gesture class assignment remains valid. To simulate corner cutting and increase sloppiness, one may optionally remove points from the resampled trajectory before one rescales. The resampling rate $N$ is critical to GPSR performance—large values result in little variation, whereas small values generate too much variation. Further, the optimal resampling rate is trajectory dependent. An optimal-$N$ solution for 2D gesture data based on gesture path density and openness was learned, but no such solution exists for 3D data. Even for 2D data, though, we are concerned with replicating score distributions rather than gesture property (feature) distributions. For these reasons, we require a new equation.

*3.2.2 Device Agnostic GPSR.* We next describe our approach to finding a new device-agnostic optimal-$N$ equation, our optimization procedure, and the results.

**Approach**: We sought to find an optimal resampling rate based on trajectory properties such that synthesized samples reproduce the within class variance when measured against their seed trajectories. We are interested in properties like path length and density that are easy to understand and calculate. In the end, we considered a number of properties based on prior gesture recognition and synthesis work [3, 39, 42], including:

(1) *Traversed Angle*: Summation of all angles (in radians) formed between consecutive vectors along the gesture path.
(2) *Sameness*: Summation of dot products between consecutive normalized vectors. Similar to traversed angle, except without an arccos conversion. We believe dot products are likely to be more robust to noise.
(3) *Density*: Length of the gesture path divided by its diagonal length. A more dense gesture is likely to have more variability.
(4) *Sharpness*: Same as traversed angle, except angles are squared, giving emphasis to corners and cusps, which may drive up variability.
(5) *Inverse Sharpness*: Sharpness result inverted, giving emphasis to straight lines.
(6) *Speed*: Duration of gesture in seconds.
(7) *Inverse Speed*: Speed result inverted.
(8) *Signal-To-Noise Ratio*: A measure of what data remains after a low pass filter is applied to a trajectory.
(9) *DP Count*: We apply angular Douglas-Peucker (DP) resampling [43] and count the number of points. We had anticipated a relationship between variability and the number of points needed to adequately describe a gesture, which the angular DP count may indicate.

Other properties we considered either violated our design criteria or were thought to be irrelevant because, although they were appropriate for statistical analysis and classification, they would not make good optimal $N$ predictors. Although we started with those properties listed above, we intended to investigate further if required, but found it unnecessary.

**Optimization**: We now describe our optimization procedure. First, for each high-activity dataset training sample described in Section 4.1, we find the within class distribution. Specifically, for a given participant and training sample, we measure its distance against each remaining within class sample. Since our dataset contains five training samples per gesture class, we take four measurements per training sample. These are combined into a single within class variance result called the target. We then perform a binary search over GPSR's resampling rate space to find $N$ whose distribution matches the target's variability. We finally then selected $N$ that minimizes the synthetic and real within class variance difference.

Once we had $N$ for each training sample, we aggregated all data together and performed stepwise linear regression to find which trajectory properties most influenced the optimal resampling rate. Analysis revealed that density and traversed angle were good candidate properties to exploit. We thereafter set out to find a solution for the following equation:

$$n = x_0 + x_1 \cdot \text{density} + x_2 \cdot \text{ angle } + x_3 \cdot \text{density} \cdot \text{angle} \quad (5)$$

We used Luus-Jaakola (LJ) optimization [31] to stochasticly search the local parameter space. Rather than attempt to fit our model to optimal $N$ values per individual gesture, we instead optimize our model to reproduce the within score distribution. (Recall that our objective is to generate a representative score distribution, not to synthesize realistic samples.) In each LJ step, we first randomly sample the local parameter space to generate new candidate coefficients. We then generate the real and synthetic within class score distributions and measure their statistical distance. Since the initial distributions are likely far apart, we begin with Wasserstein until they overlap, after which we switch to the Kolmogorov-Smirnov distance.

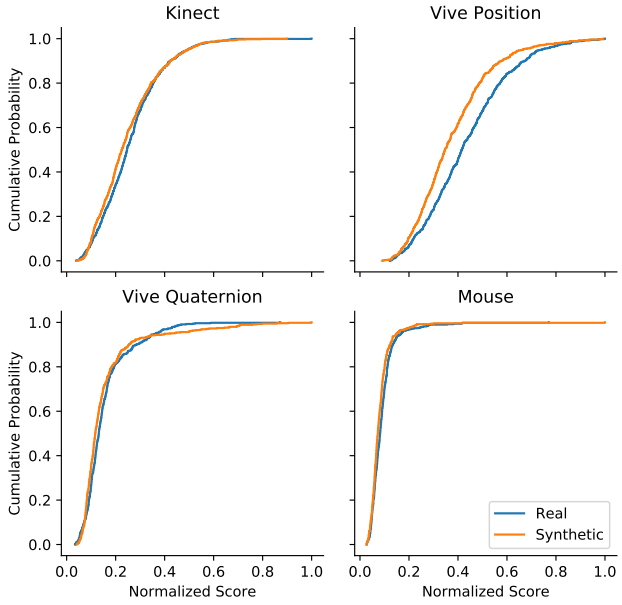


**Figure 2: Cumulative probability distribution of the within class scores measured by Jackknife for real and GPSR synthesized gestures. Results indicate by their similarity that GPSR is able to replicate the real score distribution using synthesis, especially for Kinect, Vive Quaternion, and Mouse data. Vive Position is less accurate as can be seen by the difference in distributions, though performance does not suffer as shown in our evaluation (Section 4).**

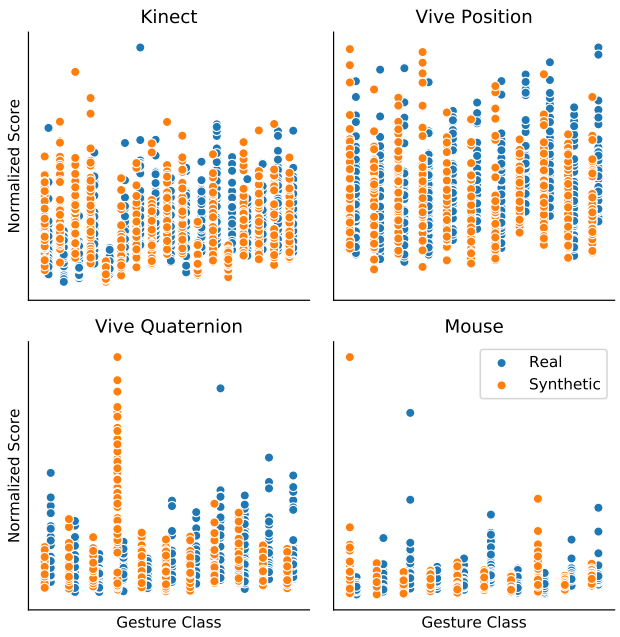


**Figure 3: Mean within class score of each training sample measured against every other sample (blue) and of each training sample measured against an equal number of synthetically generated samples (orange). Gesture classes are enumerated and organized along the x-axis so that real and synthetic distributions are collocated their class. Similarly densities indicate that GPSR is able to replicate the real gesture class distributions. Results show that the synthetic and real distributions are similar for most gesture classes, though it is clear that further improvements are possible.**

**Results**: This process produced a unique set of coefficients per input device type. The resulting cumulative probability distributions are shown in Figure 2. Although all distributions are statistically different because of sharp rises that result in moderate KS differences (between .08 to .17), we see that within their respective domains, the distributions are relatively close. One exception is with Vive Position data where it can be seen that the distributions diverge, though the divergence does not degrade performance as shown in our evaluation. In Figure 3, we plot the within class score of every training sample from every dataset. Visual inspection of location and variance reveals that GPSR is able to produce scores that reasonably replicate real gestures. Still, some classes in particular have poor synthetic results. For example, synthetic fly-like-an-eagle[2] (forth column) scores are significantly larger than real scores on Vive Quaternion data, whereas the synthetic left-right-left and right-left-right jab gesture scores are notably less than real scores on Kinect. To address these differences, we may require a third attribute beyond density and traversed angle that help decide optimal-$N$. However, since we primarily care about aggregate score distributions (Figure 2), we believe the current equation meets our requirements.

[2]Full-body gesture names from [43]. Fly-like-an-eagle - 3x movement with arms extended to the sides; left-right-left/right-left-right - jabs with respective hands in quick successions.

## 3.3 Negative Synthetic Data Generation

VKM uses a new technique called Mincer to generate synthetic negative data, which is inspired by Jackknife's splicing technique [44]. We first describe splicing and thereafter our approach.

**Splicing**: Jackknife [44] uses splicing to generate negative synthetic data, whereby two samples are randomly drawn from the training set, after which a random subsequence from each gesture is extracted and concatenated together to form a single trajectory. The negative sample is then scored against each training sample, and each score is added to the common distribution. We find that this splicing techniques works well to generate a representative negative distribution. This can be seen visually in Figure 4 where there is little divergence between real and synthetic distributions. However, as we show in our evaluation, spliced distributions do not lead to tight thresholds.

**Mincer**: In VKM, we take a different approach using a new technique we call Mincer. This approach generates gesture-specific negative samples by inserting subsequences of other training samples into the given gesture sample. As a result, Mincer yields negative samples that are on or near the seed sample's boundary. One final difference between Mincer and splicing is that minced samples are only measured against their seed gesture, rather than being measured against all samples.

In more detail, we spatially resample all training data to high resolution trajectories to ensure we preserve local features, after which we min-max normalize each sample. We next convert each trajectory into a set of direction vectors, so that the gesture set is defined as a set of displacements rather than a set of positions. To generate a negative sample from a seed gesture via mincing, we randomly select a prepared sample from a different gesture class as well as two indices. We then copy the associated subset of direction vectors from the selected sample into the seed sample at the same location specified by the indices. When the difference between indices is negative, we copy direction vectors in reverse order. We last perform a cumulative summation on the new vector set to convert these displacements into a set of positions. Figure 5 illustrates example mincing of a 2D gesture dataset. One will notice similarities between the seed gestures and their negative sample variations. However, as can be seen, the sketches appear to be realistic strokes, which implies Mincer is able to generate negative samples near the gesture boundary.

Initially, we believed it would suffice to uniformly sample indices from the full range of allowable values. We found, however, that width matters. If the resulting distribution contains too many minced samples with negligible modifications due to small width replacements (indices close together), the negative distribution shifts left as does the rejection threshold. Conversely, if the distribution contains too many significantly modified samples due to large width selections (indices far apart), the distribution shifts right as does the rejection threshold. Our first attempt to address this issue was to ensure the width fell within the middle third of possible values, following a Goldilocks-like assumption that we should avoid extreme widths. We later found that the ideal range depends on

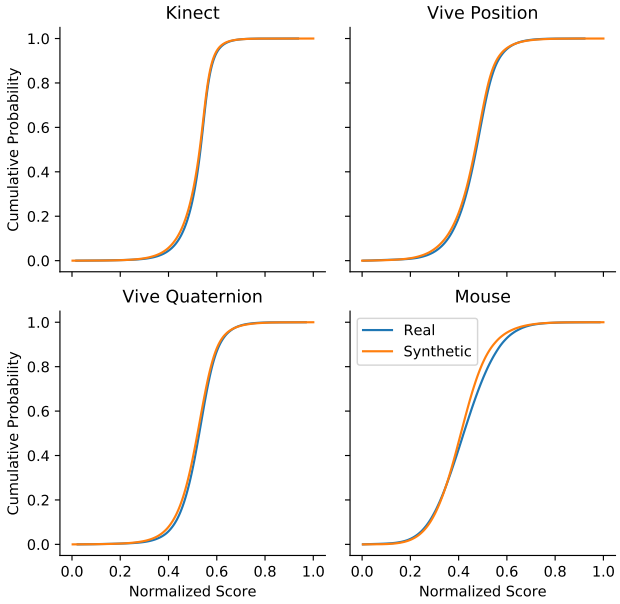


**Figure 4: Comparison of real and splice-based negative score distributions over four high-activity datasets. Measurements were made between training and real negative samples take from continuous session using Jackknife. Although the real and synthetic distributions are statistically different from each other according to two-sample Kolmogorov-Smirnov testing, we see they are sufficiently close. The issue with splicing, however, is that we must shift the distribution left in order to learn rejection thresholds that are near gesture boundaries.**

the dataset. For 2D input data, we allow widths from the lower two-thirds range, and for 3D data, we allow widths from the upper two-thirds range. It is currently unknown if this sampling strategy holds universally or must be tuned per input device or application. One possible explanation for our present finding is that because handwriting, being higher frequency data, embeds more information than full body motion, small changes in the trajectory move samples toward the gesture boundary faster than do small changes for full body gestures. We, however, leave this investigation for future work. Next, we tackle positive sample synthesis.

## 3.4 Rejection Threshold Scaling

We now address two issues that impact rejection threshold selection. First, the data collection protocol one uses to collect training data impacts gesture production variability [40]. A common data collection approach practitioners use is to collect isolated samples that users generate in a low-stress environment, one at a time, and if one is unhappy with their input, they can simply replace it. The variability of such data is likely to be less than that of the target application, where differences in environment, stress, usage, and sloppiness collectively contribute to greater variability. Since our technique attempts to locate gesture class boundaries from given

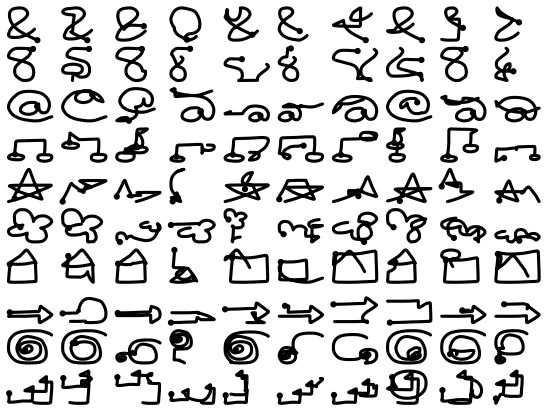

**Figure 5: Example negative samples generated with Mincer. The first column comprises training samples from the high-activity Mouse datasets (Section 4.1) followed by its subsequent minced variants. Notice how significant portions of the original seed samples appear in their negative sample counterparts yet appear to be believable sketches. This similarity allows us to better find identify gesture class boundaries when measured by a recognizer.**

training data, we propose to inflate the threshold by a value determined a priori. This **inflation factor**, unfortunately, is application dependent, and whether or not one can estimate it from training data remains unclear. How we inflate the threshold will be clarified shortly.

Second, we must reduce the threshold as the training set size increases. As illustrated in Figure 6, thresholds must be sufficiently large so as to provide adequate coverage over the application gesture class space. Note, false positives increase as we train with more data but hold the threshold constant; conversely, we are able to improve accuracy by reducing the threshold as the training set size increases. To determine how we should scale the estimated rejection threshold, we use simulation.

*3.4.1 Simulation-based Rejection Threshold Adjustment.* Our simulation accepts as input an initial threshold $\tau$, an inflation factor $\iota$, and the training set size $T$. The simulation outputs an adjusted threshold. Our goal with simulation is to find a rejection threshold scaling factor $\lambda$ that maximizes $F_\beta$. We represent the gesture class training sample space as a hypersphere centered on zero whose radius is the given rejection threshold $\tau$. The *application space* is the training space scaled up by the inflation factor $(\tau \times \iota)$. We also define a test space centered on zero whose radius is the maximum of either the application space or training sample's reach, $\max(\tau, \tau(1+\lambda))$. (A sample drawn on the training space boundary extends to $\tau(1+\lambda)$.)

To generate a test case, we draw one point from the test space and $T$ points from the training space. A test point is said to reside within the application space if its Euclidean distance from the origin is less than the inflated rejection threshold $(\tau \times \iota)$. A test point is recognized if its distance from any training point is less than the adjusted rejection threshold $\tau \times \lambda$. We classify a test point as true

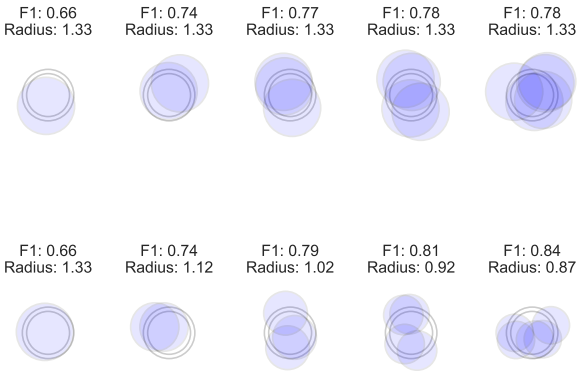


**Figure 6: Simulated effect of threshold scaling on $F_1$-score. Each subfigure comprises two rings that enclose the training and application gesture class spaces, referred to as the inner and outer rings, respectively. Individual training samples are randomly drawn from the inner space and rendered as blue circles whose radii are equal to the rejection threshold. False positive spaces are those enclosed by training samples that sit outside of the outer ring. False negatives areas are those areas within the outer ring not enclosed by a training sample. In the top row, we hold the rejection threshold constant as the training set size increases from one to five. Just below, we find the scaled rejection threshold that optimizes accuracy. Note how radial reductions lead to higher scores.**

positive if it is recognized and within the application space. A false negative point is one that falls within the application space but is not recognized. And a false positive point is one that is recognized but falls outside of the application space. To estimate an $F_\beta$-score, we generate a large number of test cases, tally classification results, and estimate accuracy from the stated results. To select an adjusted threshold, we iterate over a number of scaling factors $\lambda$ and select that which maximizes accuracy.

One problem is that of how to select the hypersphere's dimensionality. Consider that if one spatially resample 63-component Kinect trajectories to 16 points, one will then have a 1008-dimensional point. Sampling points in this space is both computationally prohibitive and unrepresentative of real gestures because it assumes complete independence between individual components. In reality, data within trajectories are highly correlated—we need only a few points to identify a gesture's class [49]. For example, Vatavu found that 2D gestures resampled to six points yield almost optimal recognition performance under Euclidean distance [49], and similar results were reported for 3D gestures [51]. We also observe via Figure 7 that the effect of dimensionality on scaling diminishes logarithmically as the dimensionality increases. For these reasons we use $d = 6$ in our evaluations, though we expect higher values will also work well.

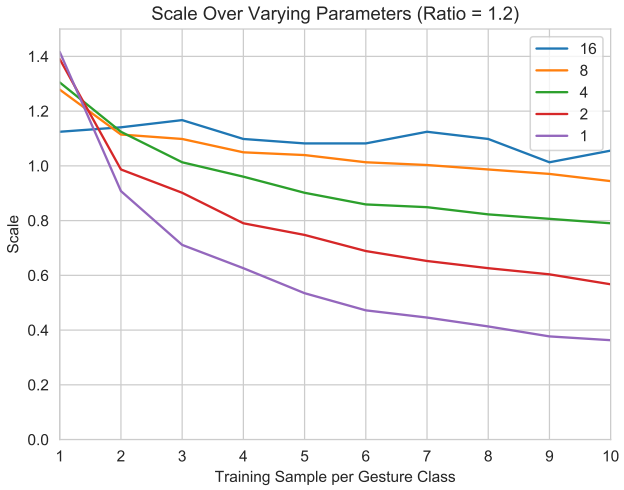


**Figure 7: Reduction in scale as training set size increases over varying dimensions. Notice that the difference in scale declines as the number of dimensions increase. These results indicate that by using simulation we can estimate a scaling function that reduces the threshold as the training set size increases.**

## 4 EVALUATION: PART I

In this section, we measure VKM's efficacy relative to alternative techniques in selecting a rejection threshold that discriminates gesture from non-gesture actions embedded in continuous high-activity data. We refer to the continuous gesture recognition pipeline used in this evaluation as the Dollar General $-family techniques. Specifically, we use Machete [43] to segment incoming continuous data into gesture candidates and Jackknife [44] equipped with the inner product measure to evaluate said candidates. Both Machete and Jackknife are $-family principled state-of-the-art, device-agnostic, and computationally efficient nearest neighbor recognition techniques. In greater detail, for each training template, Machete uses continuous dynamic programming to estimate where in time a gesture started if it ended on the current frame. Machete also estimates a similarity score; and so for each frame, Machete generates one gesture candidate per template. Using an internal heuristic to cull most gesture candidates, the majority of candidates are quickly discarded. However, since Machete is a segmenter (rather than a recognizer) its false positive rate is high. For this reason, any remaining gesture candidates are passed to Jackknife for further analysis, as was previously done [43]. Jackknife outputs a gesture recognition score for each remaining gesture candidate, which Dollar General then accepts or rejects based on the learned rejection threshold. In the remainder of this section, we describe the high-activity data, rejection thresholds selection techniques, and test protocol in more detail.

### 4.1 High Activity Data

To compare VKM against alternative rejection threshold selection techniques, we use the publicly available Machete high-activity

(HA) datasets [43]. To summarize the data, Taranta *et al.* collected five training samples per gesture per input device type tested, using ten unique participants per device: Mouse, Vive, and Kinect. Mouse data consists of 10 unistroke gestures, Vive data includes 11 gestures performed with both hands while standing, and Kinect data consists of 17 full body gestures. Vive data is further subdivided into position and orientation data so as to make four unique datasets. They thereafter collected continuous data from each participant using a Follow-The-Leader (FTL) protocol [40], whereby participants replicate the motions of a virtual actor, *i.e.*, leader, who continuously performs random actions. Their data collection tool periodically interjected gesture requests via an on screen text prompt that the participant would immediately execute, regardless of the actor's state. After gesticulating, the participant would, without pause, return to following the leader. In the end, they collected three examples of each gesture interleaved among a variety of actions that resembled puppeteering, direct object manipulations, and other non-gesture actions. Our penultimate goal is to recognize all embedded gestures with no false positives using the pipeline described above.

## 4.2 Rejection Threshold Selection Techniques

We compare several rejection threshold techniques one may use based on familiar concepts and prior work as follows:

(1) **Optimal**: This technique iterates over a range of rejection thresholds and selects that which maximizes accuracy over each user's session data, for each level of every factor. Therefore, manually labelled continuous ground truth data is used to select the optimal threshold, which serves as our maximum achievable target using the given custom gesture recognition pipeline.
(2) **Mincer**: VKM as described in the prior section whereby we use GPSR to construct a positive distribution and Mincer to construct a negative distribution, and from which we select the threshold that maximizes accuracy using simulation.
(3) $3\sigma$: Random samples are often assumed to follow a normal distribution. Under this assumption all scores will fall within three standard deviations ($3\sigma$) of their mean to include 99.7% of all samples. Even when distributions are not normal, practitioners sometimes use a $3\sigma$ pruning rule. For this reason, we consider using GPSR to estimate the positive distribution mean and standard deviation so that we may set to the rejection threshold to be $3\sigma$ above the mean.
(4) $3\sigma$-per-class: This technique is identical to $3\sigma$, except estimates and rejection thresholds are set per class, rather than globally.
(5) **Minimum Distance**: In a pairwise manner, we calculate the distance between class samples and select the minimum distance to be our rejection threshold. This is similar to constructing an N-best list [57] on training data and selecting the distance between the first and second class to be the rejection threshold.
(6) **Average distance**: If we assume that some classes partially overlap such as a square, curly, and parenthesis, then we may require a looser rejection threshold. For this reason we include the average between-class score as one option.
(7) **Splicer**: Similar to VKM except that we using the splicing technique proposed in Jackknife [44] instead of Mincer.

Note, we informally investigated additional techniques derived from those above, but none led to interesting or unique results. Since they were led by trial and error efforts rather than by informed design, we do not include them.

## 4.3 Test Procedure

We engage in user-dependent (UD) recognition testing. Our UD test procedure follows. For a given technique, dataset, participant, and training count $T$, we randomly select $T$ samples per gesture class. If there are $G$ gesture classes, then we train Jackknife with $G \times T$ samples. Thereafter, we learn a rejection threshold using the specified technique. After training is complete, we play the participant's session data through the continuous gesture recognition pipeline and record all recognition results. From hand-labeled ground truth data, we analyze classification results to generate an F1 score. This process is repeated ten times per participant and all results are averaged into a single accuracy measure.

In Section 3.2.2, we wrote that a new optimal-$N$ equation was learned from the high activity datasets used in our evaluation. It is important to note now that we use a leave one out cross validation strategy, where data from the target participant is left out of the optimization process and the resulting coefficients are in the participant's evaluation.

## 4.4 Analysis

With only 10 participants per input device, power to detect differences between multiple conditions and levels is limited. For this reason we selected the four more accurate methods (Optimal, Mincer, $3\sigma$, and $3\sigma$-per-class), and with these performed our statistical analysis, although we do report all accuracy results. Our experiment was a 2-factor within-subjects repeated measures design, where the rejection threshold selection technique and template training count were the two factors. $F_1$-score was the response measure. Because accuracy was not normally distributed, we performed ANOVA and post-hoc analysis with ART-C [12, 56]. To protect against multiple comparison type I errors, we used the Holm-Bonferroni step-down procedure [14].

## 4.5 Results

*4.5.1 Kinect.* Average accuracy for each method over varying template counts is shown in Figure 8. The rejection threshold selection method had a significant effect on overall accuracy ($F_{3,171} = 58.47$, $p < .0001$). Optimal achieved 94.1% (SD=3.3%) and was 0.7% more accurate than Mincer at 93.4% (SD=4.1%), though the difference was not significant ($t(171) = 1.15$, n.s.). However, Mincer was 4.3% more accurate than $3\sigma$-per-class at 89.4% (SD=5.3%), which was significant ($t(171) = 5.83$, $p < .0001$). Further, $3\sigma$ at only 82.9% (SD=8.9%) was 7.8% less accurate than $3\sigma$-per-class ($t(171) = -4.66$, $p < .0001$). On average, Mincer out performed all alternative methods, coming closest to Optimal performance.

The effect of template count on overall performance was not significant ($F_{4,171} = 2.02$, n.s.), but a closer look at the effect of template count on the rejection threshold selection method revealed a significant interaction ($F_{12,171} = 2.18$, $p < .05$). From one to

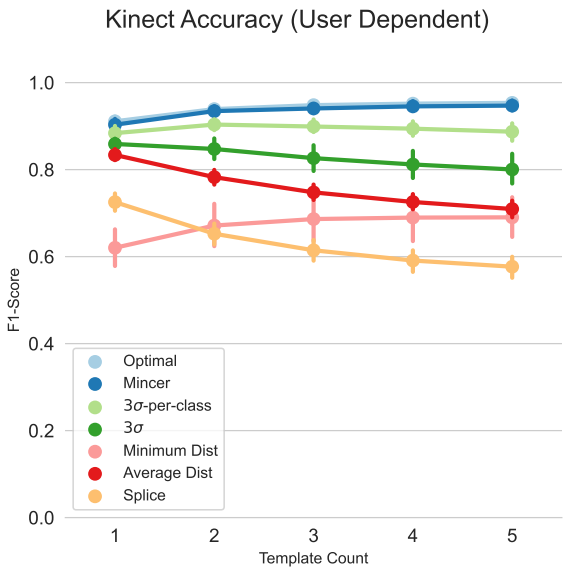


**Figure 8: $F_1$-Score over varying methods and train set sizes for Kinect user-dependent test. Error bars are standard error (68%) for legibility.**

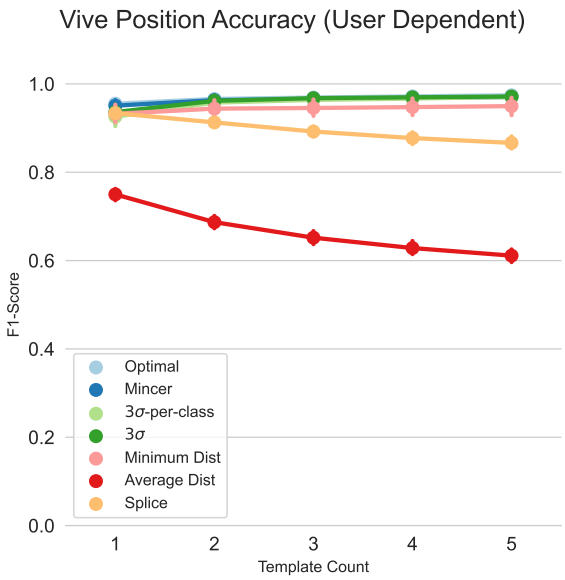


**Figure 9: $F_1$-Score over varying methods and train set sizes for Vive Position user-dependent test. Error bars are standard error (68%) for legibility.**

five templates, Optimal performance increased by 4.6% from 91.1% (SD=3.4%) to 95.3% (SD=2.7%). Mincer increased by 4.9% from 90.3% (SD=4.3%) to 94.7% (SD=3.7%). 3$\sigma$-per-class barely improved—0.4% from 88.4% (SD=5.4%) to 88.7% (SD=6.2%0). And, 3$\sigma$ saw a 6.8% decrease from 85.9% (SD=4.3%) to 80.0% (SD=10.6%). However, only the Optimal difference was significant ($t(171) = -3.75, p < .05$). It is also interesting to note that with one template loaded, the pairwise difference between each method is not significant. However, as the template count increases, Optimal is not significantly different from Mincer($t(171) = 0.51$, n.s.), but 3$\sigma$-per-class is 6.3% less accurate than Mincer ($t(171) = 4.34, p < .005$), and 3$\sigma$ is similarly 15.5% less accurate ($t(171) = 6.85, p < .0001$).

*4.5.2 Vive Position.* Average Vive Position accuracy results for each method over varying template counts are shown in Figure 9. All rejection threshold selection methods performed well. Optimal achieved a 96.7% (SD=3.1%) overall accuracy, followed by Mincer at a 96.4% (SD=3.3%), a small 0.4% reduction. 3$\sigma$ achieved a 96.1% (SD=3.8%) overall accuracy, which was 0.2% less accurate than Mincer, and 3$\sigma$-per-class achieved the lowest score of 95.7% (SD=4.3%), a 0.7% decrease from Mincer's results. However, their difference was not significant ($F_{3,171} = 2.59$, n.s.)

The effect of template count on overall accuracy was significant ($F_{4,171} = 11.97, p < .0001$), but interaction effects between the selection methods and template counts were not ($F_{12,171} = 0.43$, n.s.). Increasing template count from one to five increases the Optimal score by 2.0% from 95.5% (SD=3.3%) to 97.4% (SD=2.5%). Mincer scores improved by 2.1% from 95.1% (SD=3.8%) to 97.0% (SD=2.8%). 3$\sigma$ improvements were significant($t(171) = -3.95, p < .05$)—a 3.8% increase from 93.6% (SD=5.3%) to 97.1% (SD=2.8%). 3$\sigma$-per-class score improvements were also statistically significant ($t(171) = -4.39$, $p < .005$), increasing by 4.8% from 92.6% (SD=6.0%) at one template to 97.1% (SD=2.8%) at five templates.

*4.5.3 Vive Quaternion.* Average accuracy results for Vive Quaternion dataset for each method over varying template counts are shown in Figure 10. One will notice that overall accuracy across all methods are lower with this data type than on other datasets. The effect of the rejection threshold selection method on accuracy was statistically significant ($F_{3,152} = 45.35, p < .0001$). Optimal achieved an 89.3% (SD=4.2%) overall accuracy, followed by Mincer at 87.5% (SD=5.6%). This difference was not significant. Mincer was also 15.9% more accurate than 3$\sigma$ ($t(152) = 8.59, p < .0001$), which only achieved 73.6% (SD=10.8%) overall accuracy. And the slight lead of 1.3% by Mincer over 3$\sigma$-per-class was not statistically significant ($t(152) = 0.80$, n.s.). Optimal was 17.6% more accurate than 3$\sigma$ ($t(152) = 10.99, p < .0001$), and 3.3% more accurate than 3$\sigma$-per-class ($t(152) = 3.20, p < .01$).

The effect of the template count on accuracy was statistically significant ($F_{4,152} = 3.79, p < .01$), but there was no significant interaction between the method × template count conditions ($F_{12,152} = 1.54$). With one template loaded, none of the differences between methods were significant, and simply increasing template count for the selection methods did not result in significantly different ac curacies. Still, with five templates loaded, all methods except for 3$\sigma$ performed at around 90%. Optimal achieved 91.7% (SD=2.5%), followed by 3$\sigma$-per-class at 90.0% (SD=3.9%), and Mincer at 89.7% (SD=5.1%). Mincer was 2.2% below Optimal, and only 0.3% below 3$\sigma$-per-class.

*4.5.4 Mouse.* Average Mouse accuracy results for each method over varying template counts are shown in Figure 11. Optimal and Mincer both achieve high accuracy (above 90%) using just one

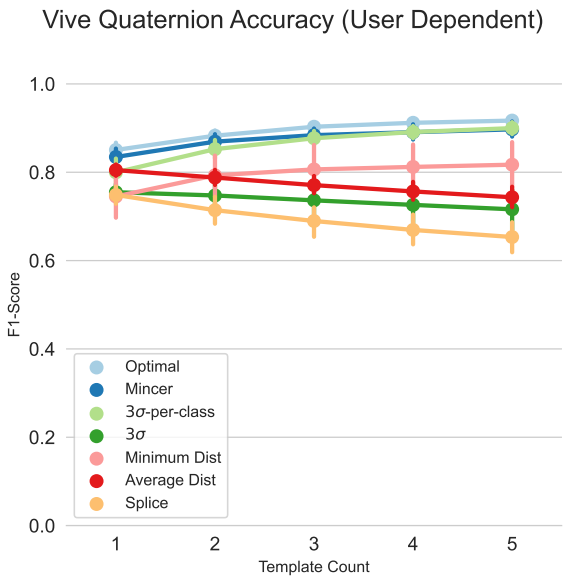


**Figure 10: $F_1$-Score over varying methods and train set sizes for Vive Quaternion user-dependent test. Error bars are standard error (68%) for legibility.**

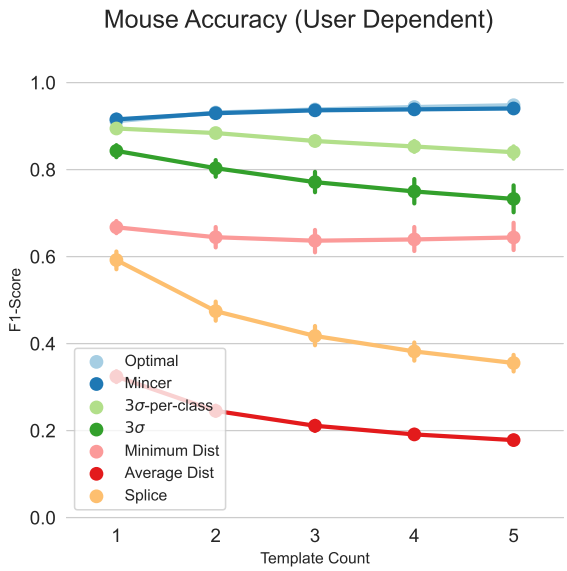


**Figure 11: $F_1$-Score over varying methods and train set sizes for Mouse user-dependent test. Error bars are standard error (68%) for legibility.**

template per gesture class, while the other methods do not achieve high accuracy. The effect of the rejection threshold selection method on accuracy was statistically significant ($F_{3,171} = 243.91, p < .0001$), with Mincer attaining 93.2% (SD=2.8%) overall accuracy, being only 0.3% less accurate ($t(171) = 0.92$, n.s.) than the Optimal at 93.5% (SD=3.0%). Mincer was also 16.3% more accurate ($t(171) = 21.80$, $p < .0001$) than 3$\sigma$ 78.0% (SD=8.3%), and a statistically significant difference of 6.9% ($t(171) = 13.82, p < .0001$) was detected between Mincer and 3$\sigma$-per-class 86.8% (SD=4.0%).

There was also an effect of method × template count on accuracy ($F_{12,171} = 6.34, p < .0001$). For instance, the Optimal score from 91.1% (SD=3.1%) to 94.8% (SD=2.2%), which was a 4.0% increase ($t(171) = -4.87, p < .0005$). A 2.7% increase was also observed with Mincer, from 91.6% (SD=3.2%) to 94.1% (SD=2.3%), but the difference was not significant ($t(171) = -3.14$, n.s.). 3$\sigma$ with one template achieved 84.3% (SD=4.3%), and 73.3% (SD=9.1%) with five templates, which was a 13.1% decrease ($t(171) = 3.36$, n.s.). 3$\sigma$-per-class scores decreased from 89.5% (SD=2.6%) to 84.0% (SD=4.6%), which was a 6.1% difference ($t(171) = 3.7705$). With one loaded template, Mincer was not significantly different from Optimal ($t(171) = -0.55$, n.s.). 3$\sigma$-per-class performed similarly, 89.5% (SD=2.6%), which was a 1.8% decrease from the Optimal score ($t(171) = 1.77$, n.s.). Accuracy was 2.3% lower when using 3$\sigma$-per-class compared to Mincer, but the difference was not significant ($t(171) = 2.32$, n.s.). With five templates loaded, Mincer achieved 94.1% (SD=2.3%) remaining close to Optimal performance—an 0.8% difference ($t(171) = 1.18$, n.s.). 3$\sigma$-per-class achieved only 84.0% (SD=4.6%), and 3$\sigma$'s accuracy was even lower at 73.3% (SD=9.1%). A significant 10.7% difference between Mincer and 3$\sigma$-per-class was detected($t(171) = 9.23, p < .0001$), as well as a 22.1% difference between Mincer and 3$\sigma$ ($t(171) = 12.59, p < .0001$). We see that Mincer is accurate with both low and high template counts, and remains close to Optimal performance through all levels, whereas other methods perform worse and drop in performance as the template count increases.

## 4.6 User Independent Testing

We also performed a preliminary investigation to understand user-independent recognition performance. Although the test protocol is similar, we vary both the training participant count and training samples per participant, in accordance with prior work [55]. All techniques, including Optimal, were less accurate. We noticed that as the training set size increased, several (in some cases all) alternative techniques dropped in performance. Mincer, on the other hand, maintained or improved in accuracy as the training set grew. For example, from one training participant with one template loaded per participant to four participants with four templates, optimal Kinect performance increased from 79.3% to 89.0%, Mincer from 73.0% to 88.1%, and, the next best 3$\sigma$-per-class, from 74.3% to 81.5%. Optimal Vive Position accuracy increased from 84.5% to 94.0%, Mincer from 73% to 93.7%, and 3$\sigma$ from 50.5% to 91.9%. Similarly, optimal Vive Quaternion increased from 57.0% to 77.7%, Mincer from 45.8% to 75.5%, and 3$\sigma$-per-class from 39.9% to 76.9%. Finally, optimal Mouse accuracy increased from 86.5% to 92.7%, Mincer from 85.1% to 87.3%, and decreased for 3$\sigma$-per-class from 84.7% to 69.2%. To fully understand the ramifications of our evaluated methods in the user-independent setting, a more rigorous investigation is required, which we leave to future work.

# 5 EVALUATION PART II

As we saw in our prior evaluation, Dollar General using VKM for rejection threshold selection was a top performer in recognizing custom gestures embedded in continuous high-activity data. In this

section we further evaluate VKM using the same Dollar General pipeline to better understand how a $-family based system performs relative to more complex continuous gesture recognition systems. For simplicity, we refer to this pipeline as The Dollar General (TDG) as its parts are derived from various $-family ideas and techniques.

## 5.1 The Eurographics 2019 Shape Retrieval Contest, Gesture Track

To understand how TDG performs relative to other continuous gesture recognition techniques, we first turn our attention to the Eurographics 2019 SHape Retrieval Contest (SHREC) track on online gesture recognition [6]. The competition organizers collected continuous data from participants that were similar to our high-activity data—their data also combines gesture with non-gesture actions. Specifically, they developed a virtual environment for the Oculus Rift that sampled hand pose data using a Leap Motion device, where "actions consisted of selecting objects, clicking on virtual buttons, moving a slider and spinning a globe with a swipe" [6, p. 94]. While interacting with the environment, participants were asked to perform one of five gestures at various times: cross, V-mark, caret, square, or circle.

The organizers collected data from thirteen participants such that gestures were performed in different positions. They then divided the dataset into a 4/9 split, where four participants were used for training and nine for user-independent recognition testing. Session data was further divided into 60 annotated training sequences and 135 test sequences. This is not strictly a customization scenario, but nonetheless represents a realistic use case where a designer acquires approximately three samples per gesture class from four peers or friends during an iterative development cycle. This is a challenging problem because, even in aggregate, the amount of training data is still relatively little, and the recognizer is trained and tested with different users.

Five groups participated in the competition, though one group was the organizers who provided a baseline recognizer based on 3 cent [7]. The other four groups provided neural network based solutions. Test sessions were played through trained recognizers that output recognition results. Post processing scripts analyzed the output and generated several error measures, including the percentage of correctly classified, mislabeled, false positive, and false negative gestures. Note that mislabeled gestures are a special kind of false positive that occurs when a gesture is detected within the window of an expected gesture, but the class label is incorrect. We ran the same evaluation on TDG using Vive Position GPSR parameters, since both involve hand tracking data.

We combine TDG results with the original competition results in Table 1. We found that TDG outperformed all alternative techniques in classification accuracy and was the only system that achieved high accuracy ($\geq$ 90%). TDG errors were mostly due to false negatives. uDeepGRU$_2$ also performed well relative to the other methods, achieving 85.2% recognition accuracy. uDeepGRU like TDG uses synthetic data generation to augment training data, which facilitates learning from limited quantities. Other methods do not fare as well, which indicates the difficulty of this challenge. It is important to note that VKM's ability to select tight rejection threshold is a key reason why TDG performs well as it does.

**Table 1: SHREC 2019 competition results reported as percentage values. Each row sums to 100%. See [6] for a description of each continuous gesture recognizer.**

| Method | Correctly Classified | Mislabeled | False Positives | False Negatives |
|---|---|---|---|---|
| The Dollar General | 90.7 | 0.7 | 0.7 | 8.1 |
| uDeepGRU$_2$ | 85.2 | 7.4 | 3.0 | 4.4 |
| uDeepGRU$_1$ | 79.3 | 8.1 | 3.0 | 9.6 |
| uDeepGRU$_3$ | 79.3 | 8.1 | 2.2 | 10.4 |
| SW 3-cent | 75.6 | 16.3 | 2.2 | 5.9 |
| DeA | 51.9 | 18.5 | 25.2 | 4.4 |
| AJ-RN | 28.1 | 43.0 | 23.0 | 5.9 |
| PI-RN | 11.1 | 39.3 | 48.9 | 0.7 |
| Seg. LSTM$_1$ | 11.1 | 28.9 | 60.0 | 0.0 |
| Seg. LSTM$_2$ | 6.7 | 25.2 | 68.1 | 0.0 |

## 5.2 uDeepGRU Performance on High-Activity Data

Based on SHREC 2019 competition results, we decided to evaluate uDeepGRU on our high-activity data. We choose to use uDeepGRU because it was the second best performer in that competition, and like TDG, uDeepGRU was designed to work with a variety of input device types. Since the competition, several improvements have been made to the system, which we use in this evaluation.

*5.2.1 uDeepGRU.* uDeepGRU is based on DeepGRU [32], a device-agnostic gesture recognizer that was shown to outperform many state-of-the-art recognizers across a variety of publicly available datasets (using test protocols adopted by the community that are unique to each dataset). In cases where DeepGRU did not achieve top performance, it still performed competitively. uDeepGRU uses a similar architecture that comprises an encoder and classification neural network. The encoder network uses unidirectional gated recurrent units [9] for its recurrent layers, and a fully connected layer for its classification network. Details are available in [6].

At each time step, uDeepGRU outputs a class label, one of which may be none, *i.e.*, no gesture detected. The system will try to learn all parts of a gesture so that it can output appropriate labels early in a gesture sequence. This is problematic with high-activity data because many non-gesture actions partially overlap with valid gestures, and without sufficient negative training data, uDeepGRU cannot learn how to separate gestures from other actions. SHREC 2019 training samples contained both gesture and non-gesture motions from which uDeepGRU learned, whereas our high-activity training dataset contains only segmented gestures. We overcome this issue in two ways. First, similar to splicing, we concatenate partial, randomly selected, gesture subsequences to both sides of full gesture samples for training. Second, all frames are labeled none, except for the last second of each real gesture sequence. These modifications help protect uDeepGRU from reporting early detections.

*5.2.2 Method and Results.* We used an identical test protocol to that reported in our first evaluation. One difference is that we used all available training samples ($T = 5$), where in each iteration, we used four random samples for training and the remaining sample for validation. After training, we replay the participant's session through uDeepGRU and save per frame label classification and score results. We say a user gesticulated if uDeepGRU outputs identical

**Table 2: uDeepGRU Kinect results**

| Method | T=1 | | T=2 | | T=3 | | T=4 | | T=5 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Optimal | 0.91 | 0.04 | 0.94 | 0.03 | 0.95 | 0.03 | 0.95 | 0.03 | 0.95 | 0.02 |
| TDG | 0.91 | 0.05 | 0.93 | 0.04 | 0.94 | 0.03 | 0.94 | 0.04 | 0.95 | 0.04 |
| uDeepGRU | — | — | — | — | — | — | — | — | 0.86 | 0.05 |

labels over a contiguous set of frames and the classification scores are above a certain threshold. Using a grid search, we find those thresholds that maximize $F_1$. Although, this is not how one would use uDeepGRU in practice, we wanted to give it the best chance possible to achieve high accuracy.

Kinect results are reported in Table 2. We find that uDeepGRU does not achieve high accuracy, even with five training samples. In fact, TDG with a single training sample outperforms uDeepGRU with five training samples. The percentage increase in error rates between optimal (baseline) and uDeepGRU is also large ($\approx$ 366%) compared to TDG ($\approx$ 9%).

## 6 DISCUSSION

It is clear from our first evaluation that The Voight-Kampff Machine is well suited to rejection threshold selection for custom gestures. Among the six threshold selection techniques tested, VKM outperformed all other selection techniques and was the only method to achieve high accuracy ($\geq$ 90%) across all four high-activity datasets in our user dependent recognition test. In three cases, we saw high accuracy even when only one training sample per gesture class was loaded, and, in all cases, performance improved with more training data. This means that designers can feel confident using VKM for rejection threshold selection for a variety of input device types.

We further found that VKM with Machete and Jacknife (TDG) also performed well on continuous high-activity hand gesture data collected with a Leap Motion input device. Specifically, TDG outperformed all competitors in the SHREC'19 competition, being the only system to achieve high accuracy. This was only possible because VKM is able to select near optimal rejection thresholds. Our competitors used state-of-the-art techniques in neural networks, but only uDeepGRU came close to our level of performance. The organizers give a possible reason, stating that "these methods require a relevant effort for optimizing training strategies working with a limited number of examples and participants had a limited time to prepare the submission. [...] other methods (DeA, AJ-RN, PI-RN, Seg LSTM) are detecting a lot of false positives, and typically provide a false detection as the first result. Proper tuning of the method could avoid this effect" [6, p. 98]. Although TDG has tunable parameters throughout the system, there is little effort involved in selecting appropriate values given that VKM is fully automatic.

It is also worth noting that uDeepGRU took over one hour to train due to its internally generated synthetic data, and because of uDeepGRU's architecture, it already trains faster than other neural network solutions. Our C++ based TDG code, on the other hand, takes less than thirty seconds to train on a 2.3 GHz Intel Core i5 based MacBook Pro with five templates loaded. Customization requires fast, online training so that users can iteratively adjust their preferences and practitioners can quickly iterate their designs. This makes TDG an ideal choice for customizable user interfaces generally, and VKM ideal for rejection threshold selection specifically.

It is interesting to find that although we learned optimal-*N* GPSR equation coefficients for specific input devices, there is some transference. We ran VKM with parameters learned from Vive Position data on SHREC'19 competition data with great success; and as noted, not only did we achieve high performance, but we also outperformed all competitors. This is not always the case, as informal testing has already revealed that we cannot use parameters learned for Kinect on Mouse input data. However, it may be that Kinect-based coefficients work well with other skeletal tracking systems designed for full body gestures, that Vive Quaternion parameters work well with other 3D orientation tracking systems, and so forth.

### 6.1 Limitations and Future Work

An issue with our current positive synthetic data generation approach using GPSR requires that we learn unique parameters for every input device. This is problematic in that if the parameters do not transfer as discussed above, designers will repeat the process we used to find suitable parameters, which is not only inconvenient but also time consuming. Even if we were able to establish a large catalog of coefficients for different input device types, HCI researchers continuously prototype new hardware and explore the usability of these devices, which often involves gesture input.

It may be that optimal-*N* depends on the unique proclivities of an input device, those characteristics that contribute to noise, pose estimation, measurement error, and constraints that influence how users move or interact with the device. Or it may be that we have not yet found the right set of trajectory attributes to examine. For example, the set of parameters that are appropriate for touch input may also be suitable for Kinect input if we were able to take into account frequency domain information and make necessary transformations. To further improve GPSR, we will have to investigate these possibilities.

A second issue is that threshold scaling requires possibly unknown information. Through simulation we determine an appropriate scaling value that takes into account differences in training and application gesture production variabilities as well as the effect of the training dataset size. Differences in variability cannot always be known in advance but is highly relevant since ignoring these differences can lead to VKM underestimating the rejection threshold, which in turn leads to an increase in false negatives. With respect to training set size, we assume that all gesture samples come from the same underlying distribution, but it may be that users use different forms of a gesture depending on context or happenstance. A straight arrow in one moment may be a curved arrow in the next, but either way, it may be a mistake to assume each instance derives from the same internalized action plan. In this case, it would be

more appropriate for simulation to assume unique distributions for each gesture if this information were available a priori. To address both of these concerns, it will be worth investigating online methods that can analyze motion data as it arrives to determine if there are deviations from the underlying assumptions.

## 7 CONCLUSION

We have introduced The Voight-Kampff Machine to solve the automatic rejection threshold selection problem for custom gesture recognition. VKM provides a general framework for threshold selection based on synthetic data generation using GPSR and Mincer. We defined a new optimal-$N$ equation for GPSR that replicates global within class score distributions. We were also able to show that Jackknife's splicing technique generates realistic negative score distributions, but this negative data generation method does not lead to optimal results. We addressed this issue with Mincer, which generates gesture class-specific negative samples, leading to better rejection thresholds. Our evaluation of four high-activity datasets revealed that TDG with VKM is able to achieve not only high accuracy, but also near optimal performance. Further, we found that our system is competitive with alternative deep learning methods despite being significantly less sophisticated. For these reasons, we believe VKM has great potential to facilitate custom gesture recognition and enable individuals to explore user interface design in ways they would not have been able to otherwise.

## ACKNOWLEDGMENTS

This work is supported in part by NSF Award IIS-1917728, Northrup Grumman, and the Department of Veterans Affairs. We also thank the anonymous reviewers for their insightful feedback and the ISUE lab members for their support.

## REFERENCES

[1] Lisa Anthony and Jacob O Wobbrock. 2010. A Lightweight Multistroke Recognizer for User Interface Prototypes. In *Proceedings of Graphics Interface 2010 (GI '10)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 245–252.

[2] Lisa Anthony and Jacob O Wobbrock. 2012. $N-protractor: A Fast and Accurate Multistroke Recognizer. In *Proceedings of Graphics Interface 2012 (GI '12)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 117–120.

[3] Rachel Blagojevic, Samuel Hsiao-Heng Chang, and Beryl Plimmer. 2010. The Power of Automatic Feature Selection: Rubine on Steroids. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium (SBIM '10)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 79–86.

[4] Lee W Campbell and Aaron F Bobick. 1995. Recognition of human body motion using phase space constraints. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*. IEEE, 624–630.

[5] Xiang Cao and Shumin Zhai. 2007. Modeling Human Performance of Pen Stroke Gestures. In *CHI Conference*. IBM Almaden Research Center and University of Toronto, 1495–1504.

[6] FM Caputo, S Burato, G Pavan, T Voillemin, H Wannous, JP Vandeborre, M Maghoumi, EM Taranta, A Razmjoo, JJ LaViola Jr, et al. 2019. SHREC 2019: Online gesture recognition. In *Workshop 3D Object Retrieva*.

[7] Fabio Marco Caputo, Pietro Prebianca, Alessandro Carcangiu, Lucio D Spano, and Andrea Giachetti. 2017. A 3 Cent Recognizer: Simple and Effective Retrieval and Classification of Mid-air Gestures from Single 3D Traces. *Smart Tools and Apps for Graphics. Eurographics Association* (2017).

[8] Salman Cheema, Michael Hoffman, and Joseph J LaViola Jr. 2013. 3D gesture classification with linear acceleration and angular velocity sensing devices for video games. *Entertainment Computing* 4, 1 (2013), 11–24.

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[10] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. 2019. AutoAugment: Learning Augmentation Policies from Data. arXiv:1805.09501 [cs.CV]

[11] Hoang Anh Dau, Diego Furtado Silva, Francois Petitjean, Germain Forestier, Anthony Bagnall, and Eamonn Keogh. 2017. Judicious setting of Dynamic Time Warping's window width allows more accurate classification of time series. In *Proceedings - 2017 IEEE International Conference on Big Data*, Jian-Yun Nie, Zoran Obradovic, Toyotaro Suzumura, Rumi Ghosh, Raghunath Nambiar, Chonggang Wang, Hui Zang, Ricardo Baeza-Yates, Xiaohua Hu, Jeremy Kepner, Alfredo Cuzzocrea, Jian Tang, and Masashi Toyoda (Eds.). IEEE, Institute of Electrical and Electronics Engineers, United States of America, 917–922. https://doi.org/10.1109/BigData.2017.8258009 IEEE International Conference on Big Data (Big Data) 2017, IEEE BigData 2017 ; Conference date: 11-12-2017 Through 14-12-2017.

[12] Lisa A. Elkin, Matthew Kay, James J. Higgins, and Jacob O. Wobbrock. 2021. *An Aligned Rank Transform Procedure for Multifactor Contrast Tests*. Association for Computing Machinery, New York, NY, USA, 754–768. https://doi.org/10.1145/3472749.3474784

[13] J Herold and T F Stahovich. 2012. The 1\textsuperscript{\textcent} 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)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 39–46.

[14] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[15] Yoonho Hwang and Hee-kap Ahn. 2011. Convergent Bounds on the Euclidean Distance. In *Advances in Neural Information Processing Systems 24*, J Shawe-taylor, R.s. Zemel, P Bartlett, F.c.n. Pereira, and K.q. Weinberger (Eds.). 388–396.

[16] Yoonho Hwang, Bohyung Han, and Hee-Kap Ahn. 2012. A fast nearest neighbor search algorithm by nonlinear embedding. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3053–3060.

[17] Kanav Kahol, Priyamvada Tripathi, and Sethuraman Panchanathan. 2004. Automated gesture segmentation from dance sequences. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*. IEEE, 883–888.

[18] Guoliang Kang, Xuanyi Dong, Liang Zheng, and Yi Yang. 2017. PatchShuffle Regularization. arXiv:1707.07103 [cs.CV]

[19] Hyun Kang, Chang Woo Lee, and Keechul Jung. 2004. Recognition-based gesture spotting in video games. *Pattern Recognition Letters* 25, 15 (2004), 1701–1714.

[20] Woojin Kang, In-Taek Jung, DaeHo Lee, and Jin-Hyuk Hong. 2021. Styling Words: A Simple and Natural Way to Increase Variability in Training Data Collection for Gesture Recognition. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 318, 12 pages. https://doi.org/10.1145/3411764.3445457

[21] Keiko Katsuragawa, Ankit Kamal, Qi Feng Liu, Matei Negulescu, and Edward Lank. 2019. Bi-Level Thresholding: Analyzing the Effect of Repeated Errors in Gesture Input. *ACM Trans. Interact. Intell. Syst.* 9, 2–3, Article 15 (April 2019), 30 pages. https://doi.org/10.1145/3181672

[22] Sven Kratz and Michael Rohs. 2010. The $3 recognizer: simple 3D gesture recognition on mobile devices. In *Proceedings of the 15th international conference on Intelligent user interfaces*. ACM, 419–420.

[23] Sven Kratz and Michael Rohs. 2011. Protractor3D: a closed-form solution to rotation-invariant 3D gestures. In *Proceedings of the 16th international conference on Intelligent User Interfaces*. ACM, 371–374.

[24] Luis A Leiva. 2017. Large-scale user perception of synthetic stroke gestures. In *Proceedings of the 2017 Conference on Designing Interactive Systems*. 1135–1140.

[25] Luis A Leiva, Daniel Martín-Albo, and Réjean Plamondon. 2015. Gestures À Go Go: Authoring Synthetic Human-Like Stroke Gestures Using the Kinematic Theory of Rapid Movements. *ACM Trans. Intell. Syst. Technol.* 7, 2 (nov 2015), 15:1—-15:29.

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

[27] Laurence Likforman-Sulem, Anna Esposito, Marcos Faundez-Zanuy, Stéphan Clémençon, and Gennaro Cordasco. 2017. EMOTHAW: A Novel Database for Emotional State Recognition From Handwriting and Drawing. In *IEEE TRANSACTIONS ON HUMAN-MACHINE SYSTEMS, VOL. 47*. IEEE, 273–284.

[28] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. 2009. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing* 5, 6 (2009), 657–675.

[29] Xiao-Hui Liu and Chin-Seng Chua. 2010. Rejection of non-meaningful activities for HMM-based activity recognition system. *Image and Vision Computing* 28, 6 (2010), 865–871.

[30] Gil Luria and Sara Rosenblum. 2010. Comparing the Handwriting Behaviours of True and False Writing with Computerized Handwriting Measures. In *Applied Cognitive Psychology, issue 24*. Department of Human Services, Haifa University,

1115–1128.

[31] Rein Luus and THI Jaakola. 1973. Optimization by direct search and systematic reduction of the size of search region. *AIChE Journal* 19, 4 (1973), 760–766.

[32] Mehran Maghoumi and Joseph J LaViola Jr. 2018. DeepGRU: Deep Gesture Recognition Utility. *CoRR* abs/1810.1 (2018). arXiv:1810.12514

[33] D Martín-Albo, R Plamondon, and E Vidal. 2015. Improving sigma-lognormal parameter extraction. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*. 286–290.

[34] Miguel A Nacenta, Yemliha Kamber, Yizhou. Qiang, and Per Ola Kristensson. 2013. Memorability of pre-designed and user-defined gesture sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1099–1108.

[35] Corey Pittman, Pamela Wisniewski, Conner Brooks, and Joseph J LaViola Jr. 2016. Multiwave: Doppler Effect Based Gesture Recognition in Multiple Dimensions. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 1729–1736.

[36] Corey R. Pittman, Eugene M. Taranta II, and Joseph J. LaViola Jr. 2016. A $-family friendly approach to prototype selection. In *International Conference on Intelligent User Interfaces, Proceedings IUI*, Vol. 07-10-Marc. https://doi.org/10.1145/2856767.2856808

[37] Réjean Plamondon. 1995. A kinematic theory of rapid human movements. *Biological cybernetics* 72, 4 (1995), 295–307.

[38] J Reaver, T F Stahovich, and J Herold. 2011. How to Make a Quick$: Using Hierarchical Clustering to Improve the Efficiency of the Dollar Recognizer. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling (SBIM '11)*. ACM, New York, NY, USA, 103–108.

[39] Dean Rubine. 1991. Specifying Gestures by Example. *SIGGRAPH Computer Graphics* 25, 4 (jul 1991), 329–337.

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

[41] Eugene M. Taranta II, Seng Lee Koh, Brian M. Williamson, Kevin P. Pfeil, Corey R. Pittman, and Joseph J. LaViola Jr. 2019. Pitch Pipe: An Automatic Low-pass Filter Calibration Technique for Pointing Tasks. In *Proceedings of Graphics Interface 2019* (Kingston, Ontario) *(GI 2019)*. Canadian Information Processing Society, 8 pages. https://doi.org/10.20380/GI2019.27

[42] Eugene M Taranta II, Mehran Maghoumi, Corey R Pittman, and Joseph J LaViola Jr. 2016. A rapid prototyping approach to synthetic data generation for improved 2D gesture recognition. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, ACM, New York, NY, USA, 873–885. https://doi.org/10.1145/2984511.2984525

[43] Eugene M. Taranta II, Corey R. Pittman, Mehran Maghoumi, Mykola Maslych, Yasmine M. Moolenaar, and Joseph J. Laviola Jr. 2021. Machete: Easy, Efficient, and Precise Continuous Custom Gesture Segmentation. *ACM Transactions on Computer-Human Interaction* 28, 1 (Jan. 2021), 5:1–5:46. https://doi.org/10.1145/3428068

[44] Eugene M Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R Pittman, and Joseph J LaViola Jr. 2017. Jackknife: A Reliable Recognizer with Few Samples and Many Modalities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 5850–5861. https://doi.org/10.1145/3025453.3026002

[45] Eugene M Taranta II, Thaddeus K Simons, Rahul Sukthankar, and Joseph J Laviola Jr. 2015. Exploring the Benefits of Context in 3D Gesture Recognition for Game-Based Virtual Environments. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5, 1 (2015), 1. https://doi.org/10.1145/2656345

[46] Eugene M Taranta II, Andrés N Vargas, and Joseph J LaViola Jr. 2016. Streamlined and accurate gesture recognition with Penny Pincher. *Computers & Graphics* 55 (2016), 130–142.

[47] Luke Taylor and Geoff Nitschke. 2017. Improving Deep Learning using Generic Data Augmentation. arXiv:1708.06020 [cs.LG]

[48] Jean Vanderdonckt, Paolo Roselli, and Jorge Luis Pérez-Medina. 2018. !FTL, an Articulation-Invariant Stroke Gesture Recognizer with Controllable Position, Scale, and Rotation Invariances. In *Proceedings of the 2018 on International Conference on Multimodal Interaction*. ACM, 125–134.

[49] Radu-Daniel Vatavu. 2011. The Effect of Sampling Rate on the Performance of Template-based Gesture Recognizers. In *Proceedings of the 13th International Conference on Multimodal Interfaces (ICMI '11)*. ACM, New York, NY, USA, 271–278.

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

[51] Radu-Daniel Vatavu. 2013. The impact of motion dimensionality and bit cardinality on the design of 3D gesture recognizers. *International Journal of Human-Computer Studies* 71, 4 (2013), 387–409.

[52] Radu-Daniel Vatavu. 2017. Improving Gesture Recognition Accuracy on Touch Screens for Users with Low Vision. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4667–4679.

[53] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. 2012. Gestures As Point Clouds: A $P Recognizer for User Interface Prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction (ICMI '12)*. ACM, New York, NY, USA, 273–280.

[54] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. 2013. Relative Accuracy Measures for Stroke Gestures. In *Proceedings of the 15th ACM on International conference on multimodal interaction (ICMI '13)*. ACM, ACM, New York, NY, USA, 279–286.

[55] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2018. $Q: A Super-Quick, Articulation-Invariant Stroke-Gesture Recognizer for Low-Resource Devices. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3229434.3229465

[56] Jacob O Wobbrock, Leah Findlater, Darren Gergle, and James J Higgins. 2011. The Aligned Rank Transform for Nonparametric Factorial Analyses Using Only Anova Procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 143–146.

[57] Jacob O Wobbrock, Andrew D Wilson, and Yang Li. 2007. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 159–168.

[58] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2020. Random Erasing Data Augmentation. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 07 (Apr. 2020), 13001–13008. https://doi.org/10.1609/aaai.v34i07.7000