THE DOLLAR GENERAL: CONTINUOUS CUSTOM GESTURE RECOGNITION TECHNIQUES AT EVERYDAY LOW PRICES

by

EUGENE M. TARANTA II
B.S. Computer Science. University of Central Florida, 2006
M.S. Computer Science. University of Central Florida, 2012

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2020

Major Professor: Joseph J. LaViola Jr.

ABSTRACT

Humans use gestures to emphasize ideas and disseminate information. Their importance is apparent in how we continuously augment social interactions with motion—gesticulating in harmony with nearly every utterance to ensure observers understand that which we wish to communicate, and their relevance has not escaped the HCI community's attention. For almost as long as computers have been able to sample human motion at the user interface boundary, software systems have been made to understand gestures as command metaphors. Customization, in particular, has great potential to improve user experience, whereby users map specific gestures to specific software functions. However, custom gesture recognition remains a challenging problem, especially when training data is limited, input is continuous, and designers who wish to use customization in their software are limited by mathematical attainment, machine learning experience, domain knowledge, or a combination thereof. Data collection, filtering, segmentation, pattern matching, synthesis, and rejection analysis are all non-trivial problems a gesture recognition system must solve.

To address these issues, we introduce The Dollar General (TDG), a complete pipeline composed of several novel continuous custom gesture recognition techniques. Specifically, TDG comprises an automatic low-pass filter tuner that we use to improve signal quality, a segmenter for identifying gesture candidates in a continuous input stream, a classifier for discriminating gesture candidates from non-gesture motions, and a synthetic data generation module we use to train the classifier. Our system achieves high recognition accuracy with as little as one or two training samples per gesture class, is largely input device agnostic, and does not require advanced mathematical knowledge to understand and implement. In this dissertation, we motivate the importance of gestures and customization, describe each pipeline component in detail, and introduce strategies for data collection and prototype selection.

This dissertation is dedicated to my mother, Catherine M. Nieto, for setting me up for success; to my wife, Mary A. Taranta, for her sacrifices, love, and support; and to our children, Jacob Matthew Taranta and Lydia Rose Taranta, for whom I hope this work inspires....

...like, literally, I expect them to become gesture recognition experts.

ACKNOWLEDGMENTS

Recalling back to my first day of *CAP 6121: 3D User Interfaces for Games and Virtual Reality* when Dr. Joseph J. LaViola Jr. opened with, "If you work really, really hard, you can earn a B," I knew I was in for a stressful, yet properly challenging and exciting ride. Though, I certainty didn't expect that the ride to extend beyond the semester and into the many years that followed, carrying me through varying degrees of uncertainty, stress, resolution, and happiness, but I am ever grateful it did! I am especially grateful to Dr. LaViola for supporting me as I pursued topics of personal interest, even when they had fallen out of fashion; for never allowing me to lower the bar, even when it would have been convenient; and for challenging me philosophically on the importance, meaning, and nature of my work. It is hard to imagine that I could have gotten more out of my time at UCF with any other advisor. Also, now that we are here, I feel I can finally call you Joe ;)

I am grateful to my committee, Dr. Jacob O. Wobbrock, Dr. Hassan Foroosh, and Dr. Annie S. Wu for their support and encouragement. I am especially grateful to Dr. Wobbrock for recognizing the importance of accessible techniques that solve practical problems and establishing a body of work off which I was able to build. It has had an impossible to measure impact on my life, and one that I believe will continue to pay out for some time to come. I also want to thank Dr. Sumanta Pattanaik for his encouragement and support early in my graduate career. I wish we could have done more together.

A big shout out goes to the Interactive Systems and User Experiences (ISUE) lab's past and present members, especially Corey Pittman, Mehran Maghoumi, Kevin Pfeil, Kate Kapalo, Brian Williamson, Andrés Vargas González, and Jack Oakley. Extra praise and thanks goes to Corey and Mehran: It's been a tremendous pleasure working with you over all these years, taking on and tackling all these little problems. You helped me out more than you can know, and the only disap-

pointing thing about being done is that our opportunities to collaborate will be less. I'll miss the interesting discussions and late night writing marathons. I also thank Mykola Maslych for his help and support right up to the very end; I'm looking forward to watching you develop as a research scientist.

As Will Smith laments about acting and rapping in *I Wish I Made That/Swagga*, "It might have stunted my growth but I wanted them both... you do two things and then they both gonna suffer," I similarly felt the tension between work and school, unintentionally sacrificing one or the other (or both) at various times. I therefore thank my previous managers at Xyratex and Seagate for their patience, understanding, and encouragement—namely John Smyder, Pranay Varma, Jeff Clegg, and Todd Roberts. A special thanks goes to Jim Valentino for his encouragement even before I returned to school. Also, thank you to my colleagues at Northrop for their recent support as I wrapped this up.

I am grateful to my parents Eugene Taranta (Dec. 2, 1958—Sep. 26, 2016) and Catherine Nieto. Especially Catherine. They provided my sisters and I with a stable environment, put us in good schools, gave us love and comfort, set high expectations for us from an early age, and gave us more than they ever had. Three children, three masters degrees and now a doctorate is the result of their effort—this is their achievement as much as it is mine. Mom and Dad, I hope we can do for our children what you have done for us. Thank you for everything!

I'm also appreciative of my sisters', Alex and Nicole, support throughout my time in graduate school, and for helping me get through many holidays by sharing gift ideas or letting me just tag along. This allowed me to focus on school and get through what were some very tough deadlines.

To my wife and best friend, Mary, whom I owe a debt that can never be repaid, thank you for believing and supporting me; for raising Jake and Lydia with abundant love, care, and respect; for taking care of the house as well as the landscaping; and, well, for taking care of almost everything,

all of which would have fallen into a disrepair without you! For reading, rereading, reading again, and then reading one last time parts of this dissertation, I again thank you. Without you in my life, this work would not mean nearly as much as it does with you in it, and the best thing about being done is being able to move on to start the next part of life with you! (And also to make some mother #@!*$% money!)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: WELCOME TO THE DOLLAR GENERAL

Welcome back my friends,
to the show that never ends

We're so glad you could attend,
come inside, come inside

Emerson, Lake & Palmer
Karn Evil 9 1st Impression, Pt 2

"H! How are you?" one might exclaim with a broad grin and enthusiastic wave, spotting an old friend standing in line at the store, whereas a mumbled hey and imperceptible nod are all that the same individual offers a passerby in the parking lot. Here, one's gestures reveal more than the words themselves, communicating dichotomous ideas and emotions that are only understood in context. Though nuanced and complex, these nonverbal communications are ingrained in us to the extent that gesticulation is thought to predate language [46]. Throughout times of classical antiquity, from Aristole to Cicero to Quintilian, gestures were analyzed, debated, and understood as being relevant in their ability to convey emotion and persuade [120]; even blind persons use them in conversation with other blind individuals, despite having no frame of reference for understanding their meaning[108]. So it is unsurprising that as computers began to escape their laboratory confines in the eighties—permeating into our homes, businesses, and schools—research in natural user interfaces has included a broad exploration of gestures as a means of interacting with computers. Over this short period of time, investigators have brought gestures to 3D scene sketching [275], chemistry [239], mathematics [113, 237], music [26], surgery [178], and physics [39], using input devices such as data gloves [70], depth cameras [21], electromyographs [264], game controllers [99], interactive tabletops [158], mice [215], pen and touch devices [267], steering wheels [6], and wearables [210].

What is surprising, however, is that many techniques are particular in how they handle recognition. Unique approaches exist for pen, finger, hand, arm, foot, and full body gesture recognition, suggesting that there are no widely known "go-to" methods that researchers, practitioners, hobbyists, and students can turn to for accessible, custom gesture recognition support. In practical terms, this problem means that if one's ambition is to develop a gesture-based user interface, one must acquire enough machine learning knowledge that he or she is able to answer a variety of tough questions. *Should I use decision trees, neural networks, random forests, or SVMs? Are random neural network forests a thing? What are features and how do I engineer them? Why is invariance important? How much data do I need to collect in order to accurately estimate a covariance matrix? How much data do I need, period?* These are the kind of uneasy questions a designer (whose primary concern is user interface design, not pattern recognition) would rather avoid. One can use a library to overcome some of these hurdles, but not all questions are resolved. One must still select an appropriate machine learning technique and understand enough about their input device, gesture set, parameter options, and domain to effectively utilize the library.

Augmented and virtual reality systems whose popularity are in resurgence only further exacerbate this problem, as XR systems operate in a space where physical interactions extend beyond simple planar motions. First, sensors tend to be less reliable, yielding noisy signals that impact recognizer accuracy. Although low-pass filters sufficiently clean noisy input, they are difficult to tune—their optimal parameter values depend on the amount of activity and noise present in a signal [230]. Second, as we move from two to three dimensions, the gesture production variabilities increase, which in part may be why good 2D recognizers find less success in this more difficult space. Third, data streams are often continuous. This means that one must spot gestures as they come along, introducing yet three more complexities: namely, one must segment the input stream; one must determine a rejection criteria that distinguishes gesture from non-gesture sequences; and one must ensure the entire process runs at interactive rates. Fourth, in order to support gesture

customization, these problems must have solutions that work with little training data. In aggregate, we see a tremendous challenge in providing the community with a generic, straightforward, reliable broad-spectrum custom gesture recognizer. We address this challenge herein, demonstrating the thesis that:

> *A suite of automatic, device-agnostic, yet practical techniques collectively called The Dollar General is able to filter, segment, and recognize complex custom gestures in a continuous input stream with high accuracy using only a few training samples per gesture class.*

Chapter 2 (Foundations) provides an in depth explanation of all keywords, but let us begin to understand the basic concepts. By *complex gesture*, we mean dynamic human motion that intentionally communicates a message; and by *continuous*, we mean patterns embedded in long sequences whose temporal end points are not delineated through hardware interactions or heuristics. To recognize complex gestures in continuous data streams, we introduce The Dollar General, a pipeline that uses a variety of *practical* techniques. These are algorithms that most upper level undergraduates can understand and write from scratch, though we believe talented lower level students will also be able to implement them. With these concepts in mind, we are now in a position to outline our pipeline.

1.1 The Dollar General

The Dollar General (TDG) is a pipeline of novel continuous custom gesture recognition techniques. As illustrated in Figure 1.1, an input device periodically measures its hardware sensors and outputs a discrete signal. This signal contains high frequency noise that we remove with a

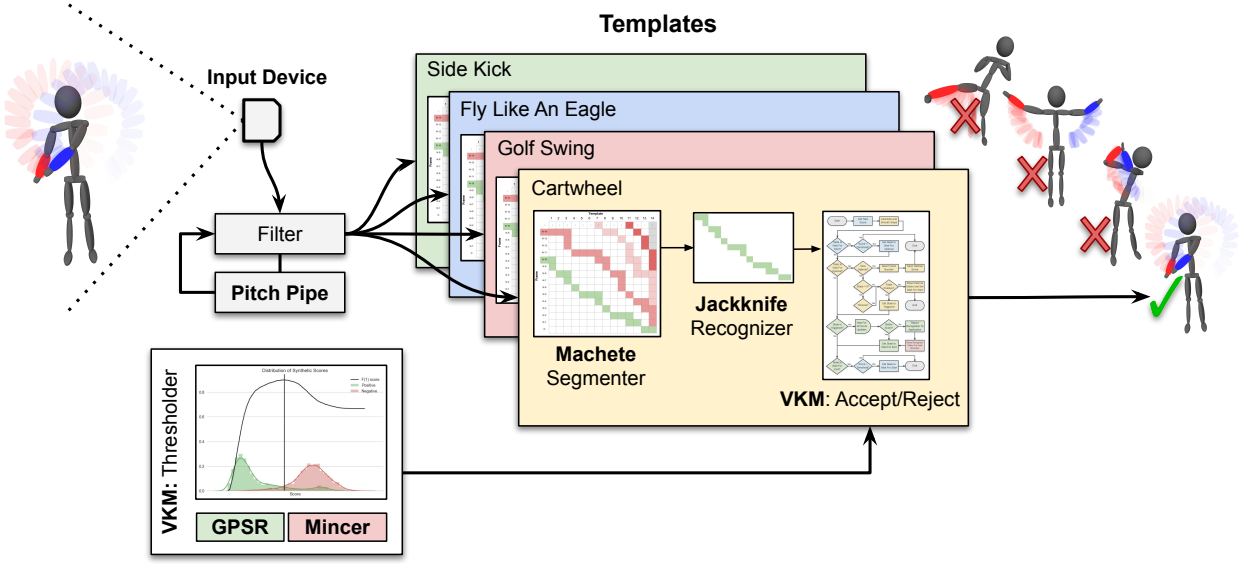


Figure 1.1: The Dollar General continuous custom gesture recognition pipeline.

low-pass filter, letting through only the most relevant information, *i.e.*, frequencies that embed human motion. **Pitch Pipe** monitors the input signal and automatically calibrates the filter according to noise and activity level measurements. The filtered signal thereafter enters the main recognition subsystem where we watch for known gesture patterns. Specifically, **Machete** segments the filtered signal into gesture candidate subsequences, and qualified candidates are passed to **Jackknife** for further evaluation. Jackknife scores each candidate and passes its results to The **Voight-Kampff Machine** (VKM), our accept-reject state machine that finally decides whether a gesture candidate is a real gesture or non-gesture action.

Within VKM's accept-reject criteria is a rejection threshold (boundary measure) that distinguishes gesture class replicas from other possible motions. VKM uses synthetic data to generate representative score distributions that it then uses to select an appropriate threshold. In more detail, VKM uses **Gesture Path Stochastic Resampling** (GPSR) to generate valid positive gesture samples and **Mincer** to generate negative (non-gesture) samples. VKM analyzes the synthetic samples using

Jackknife and selects a rejection threshold that maximizes accuracy.

1.1.1 On Naming Our Pipeline

The Dollar General (TDG) is heavily influenced by the $1 custom gesture recognizer [267] and its subsequent body of work, collectively referred to as the $-family. Thus, TDG pays homage to its inspiration by including *Dollar* in its name. Second, our pipeline embodies a number of individual techniques, any of which one may use independently. That is, Pitch Pipe, Machete, Jackknife, GPSR, Mincer, and VKM are all unique constructs that do not depend on one another. In this way, we offer a variety of useful continuous custom gesture recognition techniques via a one-stop shop solution. Dollar General Corporation is similarly a chain of variety stores found throughout the United States that sells a range of household items, hence our inclusion of *Dollar General* in TDG. Further, $-family recognizers are often touted as being low cost (in understanding and implementation effort) as are the variety of items sold in Dollar General stores, a further parallel to our pipeline. Last, TDG is internet slang for "Too Damn Good", which is, of course (even if wrong), how we feel about our work.

1.2 Contributions

This work makes the following major research contribution:

- The Dollar General — A full pipeline for continuous custom gesture recognition that is device-agnostic, comprising several novel techniques for automatic low-pass filter tuning, segmentation, dissimilarity measurement, synthesis, and rejection.

The techniques we developed for TDG in further detail are:

- Pitch Pipe — An automatic calibration technique for single parameter low-pass filters that monitors the input signal, estimates noise and activity, and selects the parameter level that best smooths the signal. This work also spawned a derivative calibration technique for pointing tasks [230].

- Penny Pincher [231, 238] — An ultra fast yet competitive nearest neighbor recognizer for 2D gestures that exploits the inner product measure on direction vectors to perform pattern matching under exceptionally tight time budgets.

- Jackknife [235] — An extension of Penny Pincher for custom 3D gestures that incorporates elastic matching via dynamic time warping (DTW). Jackknife further improves on DTW by adjusting this measure with novel correction factors that better separate gesture classes.

- Machete [233] — An end point localization technique that combines elements of Jackknife with continuous dynamic programing for fast and effective custom gesture segmentation, which can be paired with a robust underlying recognizer for gesture spotting.

- Gesture Path Stochastic Resampling (GPSR) [232] — A positive sample synthetic data generation method one can use to increase the training set size.

- Mincer [235][1] — A synthetic data generation method for creating non-gesture samples from training data.

- Voight-Kampff Machine (VKM) — An accept-reject machine that when given a dissimilarity measure decides whether or not the associated pattern is a gesture class replica. One major VKM component is the machinery that selects a rejection threshold from synthesis. VKM combines GPSR samples with Minced training data to generate representative score

[1]The predecessor to Mincer appeared in our Jackknife paper, but was unnamed.

distributions, and by analyzing these distributions, VKM is able to choose an appropriate rejection threshold.

Additional contributions related to custom gesture recognition are:

- An evaluation of two $-family principled prototype selection techniques that are shown to be effective data reduction techniques for large training set sizes [191]. Based on this work, we propose future work that incorporates random mutation climbing into continuous online retraining.
- Two new datasets we used to evaluate our techniques, a low-activity dataset for Kinect and Leap Motion, and a high-activity dataset for Kinect, HTC Vive, and mouse.
- An evaluation of three data collection techniques that demonstrate how standard practice does not produce ecologically valid training data, especially for video games [234].
- A new, easy to implement data collection protocol called Follow The Leader that increases gesture production variability relative to standard practice [234].

1.3 Reader's Guide

The Dollar General comprises several concepts and components that we organized into a number of mostly self contained chapters as follows:

Chapter 1 — Begins to motivate the need for new techniques that are suitable for recognizing custom gestures in a noisy continuous input stream and that are accessible to practitioners of varying backgrounds, skills, and knowledge.

Chapter 2 — Extends those concepts presented in the introduction and provides foundational material that covers our philosophy, nomenclature, and motivation. In doing so, we begin to cover related work, including a powerful model of human motion called the Kinematic Theory that enables us to reason about movement and gesture variability.

Chapter 3 — Builds on the foundations chapter, but presents a deeper discussion on the state-of-the-art in gesture recognition, especially from a customization perspective.

Chapter 4 — Introduces Pitch Pipe, our automatic method for tuning low-pass filter parameters.

Chapter 5 — Introduces our most streamlined recognizer, Penny Pincher.

Chapter 6 — Presents a straightforward dissimilarity measure for evaluating gesture candidates that works with a variety of input device types which we call Jackknife and is an extension of Penny Pincher.

Chapter 7 — Presents Machete, an efficient inner product based continuous dynamic programming approach for segmenting continuous input into candidate subsequences that can be further evaluated by an underlying dissimilarity measure such as Jackknife.

Chapter 8 — Presents our synthetic data generation method, GPSR, that one can use to increase the training set size.

Chapter 9 — Presents The Voight-Kampff Machine (VKM) for selecting rejection thresholds from synthetic data. This chapter also introduces Mincer for generating negative, non-gesture samples.

Chapter 10 — Provides an evaluation of The Dollar General that compares its performance against state-of-the-art continuous recognizers under varying conditions.

Chapter 11 — Discusses an evaluation of two $-family principled prototype selection techniques,

which one can use to reduce their training set size if one, over time, finds one's self with a large collection of samples.

Chapter 12 — Presents evidence that standard practice data collection procedures are not ecologically valid for video games, and introduces a new protocol called Follow The Leader, which begins to address this problem.

Chapter 13 — Discusses lessons learned and identifies future work.

Chapter 14 — Concludes the work.

Appendix A — Presents an evaluation of rejection threshold selection based on splicing, which later led to Mincer.

Appendix B — Presents pseudocode for the techniques discussed throughout this dissertation.

Appendix C — Presents copyright release and permissions information.

Appendix D — Presents our IRB approval letters.

CHAPTER 2: FOUNDATIONS

I'm the operator,
with my pocket calculator

I am adding and subtracting
I'm controlling and composing

By pressing down a special key,
it plays a little melody

Kraftwerk
Pocket Calculator

In this chapter, we establish the grounds, philosophy, language, and motivation upon which the remainder of this dissertation builds. Once we lay this foundation, one will be in a better position to appreciate the subsequent related work and design of our continuous custom gesture recognition pipeline.

2.1 Overview

One **datum** is a fact or sampling of the world's state at a particular moment in time, such as a potentiometer reading, an instance of a random variable, or the position of one's hand along a trajectory just moments into the production of a gesture. Without data, we are unable to make practical or useful claims, as it is through the relationship between facts that we come to understand the world. For example, to answer the question, *Is it safe for me to listen to my* Dark Side of Moon *vinyl at 150* dB, one must interpret the proposed volume in relation to other known facts. By recalling that whispers, restaurant noise, rock concerts, and hearing loss-inducing shuttle rocket launches respectfully resonate at 30, 70, 110, and 180 dB, we can infer that the answer to our

question is no. In this way, we ask questions of data that careful analysis will answer. What is the mean, standard deviation, and skew of these numbers, and how well do they fit a Gaussian distribution? A lognormal distribution? To what words do these phonemes map? Based on features of this photo, what were the camera's internal calibration parameters? Answers to such questions resolve uncertainty, which is to say that the output of a function on data yields **information**, and, generally, with more data comes more certainty. In this context, our objective with continuous gesture recognition is to algorithmically extract information encoded in motion signals so as to answer the following question: Given a signal at the user interface boundary, what if any command did a user intend to communicate? Part of what makes this question especially challenging is that we will try to answer it using as little data as possible in order to support end-user customization.

Before we continue, though, let us better understand the landscape. The relationship between two variables such as that between time and position is a **signal** encoding information about an underlying process. In the context of a human computer interaction, this underlying process relates to an intentional communication designed to invoke a specific software function; and when this communication is in the form of a **gesture**, the process generates a well known neuromuscular response correlated with the command that one expects their system to recognize. That is, one first decides to interact with software in accordance with a user-specific objective, *e.g.*, to make their avatar backflip in a video game. He or she then forms an **action plan**—a sequence of virtual targets connected by neuromuscular commands that in aggregate form a complex trajectory—and executes the plan. According to the **Kinematic Theory of rapid human movements** [193, 194], an action plan can be fully described by a Sigma-Lognormal model [195], which is a vectorial summation of individual lognormal primitives connecting the virtual targets. Each primitive (comprising parameters for activation time, duration, velocity, and angular position) is processed by the motor cortex which in turn actives the appropriate neuromuscular networks to generate the desired motion. Because an action plan and its activation conforms to a standard, one's articulation over

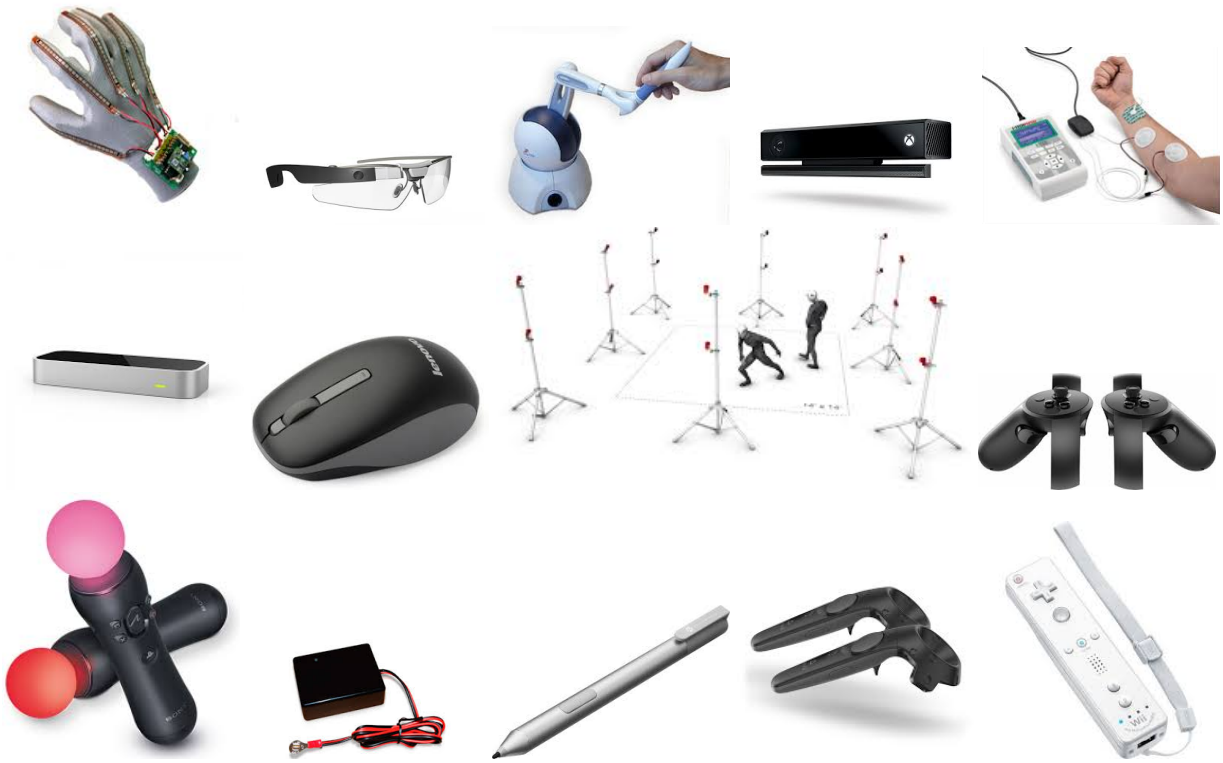


Figure 2.1: Example input devices through which users may submit gestures to a computer system. Each device samples and reports different units of information, with varying levels of noise and accuracy.

time encodes a meaningful message—the command. Consequently, most of our effort as gesture recognition researchers goes into figuring out how to interpret commands from trajectory data.

To analyze motion, though, we must first be able to capture and represent signals in software. An **input device** periodically samples specialized sensing hardware that measures the environment and renders the measurements into a discrete signal, which we sometimes call a **time series**:

$$X = (\boldsymbol{x}_i \mid i = 1 \ldots N), \tag{2.1}$$

where $|X| = N$ is the number of samples in the series, and each datum $\boldsymbol{x}_i \in \mathbb{R}^m$ is a measure of the sensor's state at time step $i \in \mathbb{N}$. We use "time step" loosely in that steps may be ordinal rather than interval in nature and not truly indicative of time. The dimensionally m of each $\boldsymbol{x}_i$ depends on the input device and what it measures. For instance, a device may measure 2D coordinates via a stylus on an interactive display, 3D joint coordinates via RGB-D data in cooperation with computer vision, or gravitational forces via an accelerometer. This is to say that one can capture motion in a variety of ways, using a variety of input devices, of which some are shown in Figure 2.1.

Regardless of the input device or data type, we must ultimately match new patterns with known gesture classes. We understand by **class** the set of all motion patterns that map to the same command. A practical software application will support any number of classes that in aggregate form the application's **gesture dictionary**:

$$\mathbb{G} = \{g_i \mid i = 1 \ldots N\}. \tag{2.2}$$

In words, each $g_i \in \mathbb{G}$ is a unique gesture class (enumerated by $i \in \mathbb{N}$), and $|\mathbb{G}| = N$ is the number of classes in the dictionary. Given an unknown trajectory, **query** Q, we reason about user intention by deriving information from Q through careful analysis, or equivalently by evaluating Q in relation to other known facts. One may pursue two complementary paths of inquiry to resolve uncertainty, which are inquiries of context and likeness. Context is useful in helping one determine intention given that probable actions are correlated with the state of affairs at the time of gesticulation, thereby justifying increased sensitivity toward certain gestures [236]. However, in this work we are concerned mostly with likeness where, given a dissimilarity measure, we calculate to what extent Q resembles each class in the dictionary. Specifically, for a given class, we have one or more mathematical **models** that describe the class's distinguishing characteristics. We then evaluate how well query Q relates to each model so as to understand in what way the query differs from each

class standard. If we find sufficient similarity, we may infer that Q is an instance of the class. How one defines models and evaluates dissimilarities is the subject of pattern recognition. While there are numerous pattern recognition approaches [59], perhaps more fundamental is how gestures are specified.

One can specify gestures by description using a high level language [131, 214] or by example. In the latter case, example motions sampled from gesticulating individuals are used to **train** class models, from which the recognizer learns one or more representations of a target class. Arguably, specification by example is far more popular than specification by description because as Rubine writes [214](p.18):

> In order to specify gestures by description, it will be necessary for the specifier to learn a description language. Conversely, in order to specify by example, the specifier need only be able to gesture. Given a system in which gestures are specified by example, the possibility arises for end users to train the system directly, either to replace the existing gestures with ones more to their liking, or to have the system improve its recognition accuracy by adapting to the particular idiosyncrasies of a given user's gestures.

Here, Rubine alludes to customization, a topic that is of central interest in modern gesture recognition research and of primary concern in our work. Further, how one can achieve high recognition accuracy with as few samples as possible is of equal importance, because in customizing a user interface, we do not want to burden the user with significant data collection work.

Beyond gesture recognition, we have segmentation. Most input devices are continuously sampled, but mechanical input device mechanisms allow users to demarcate when in time a gesture begins and ends, making at least the intent to interact unambiguous, even if software cannot recognize the given gesture. This is the case with stylus up, down events on an interactive display and trigger

toggles on a Wiimote or Oculus Rift controller. However, when triggers and buttons are mapped to unrelated functions or when a device uses passive sensors, then one must resort to alternative techniques. Two classes of continuous gesture recognition involve **spotting** and **segmentation**. The literature often uses these terms interchangeably, but we draw an important distinction. Under the former condition, we simply have to recognize that a gesture has occurred, whereas with segmentation we must also localize gesture end points. Often spotting is sufficient, but when a gesture conveys relevant spatial information (*e.g.*, "put that there"), we require precise segmentation. We also require such precision when the system relies on an underlying recognizer to classify gestures. In this case, the segmenter must isolate candidate gesture subsequences in an unbounded time series and submit the candidates to a robust recognizer for further evaluation.

Finally, from the mind of an individual and through their neuromuscular system, through the environment and to the sensor of an imperfect input device, gesticulation is subject to numerous perturbations, variations, and corruptions. Although a prototypical action plan conforms to a standard, its effectuation will fall short of perfection due to fluctuations in physical and emotional state, cognitive load, skill, and other environmental factors. We observe for instance that gesture productions differ across datasets [232], over varying articulation speeds [251], and when a user is training a game recognizer versus when he or she is playing the game [38, 236, 238]. Referring back to the Kinematic Theory, we understand that a Sigma-Lognormal primitive comprises six parameters, each following a different random distribution [161], and given that an action plan further comprises many primitives, it is unsurprising that even within homogenous conditions, their summation yields significant variability. In addition to the human element, hardware and software communication channels are subject to thermal noise and interference as well as precision and estimation errors which manifest as data corruptions that users sometimes perceive as jitter [187]. These phenomena combine to produce variations in speed, shape, scale, skew, position, duration, orientation, and noise. Therefore, to properly evaluate a given query, a recognizer must

be **invariant** to these differences so as to discover and understand a user's intention. How to overcome this challenge is in part the subject of this work as it is with every work on segmentation and recognition. However, in solving these problems, we further constrain ourselves to $-family [8, 9, 143, 250, 253, 267] principles, which above all, demands simplicity in a way that does not sacrifice quality.

2.2 Gestures and How They Differ From Other Things People Do

Gestures are a type of signal. We say that the dependency of one variable on another is a signal when their relationship conveys information about an underlying process. Thus, by analyzing signals, we hope to extract embedded information so as to learn something about the process that generated it. For example, speech-induced vocal chord vibrations render a signal of varying air pressure over time, encoding phonemes of a spoken language that we hear and understand as words. In human-computer interactions (HCI) *neuromuscular* and *physiological* signals are of particular interest. To the latter, brain computer interfaces (BCI) depend on electroencephalography (EEG) signals that measure voltage fluctuations due to brain activity. Since EEG patterns vary over different mental tasks, users may control a computer by imagining actions, visualizing objects, or performing computations that the system then in turn recognizes as a command [14]. In combination with neuromuscular activities, physiological signals such as heart rate and respiration often convey information about one's internal state, *e.g.*, their affective and attitudinal mood, and it is the goal of behavior signal processing (BSP) to extract this information [170], which we may then, among other things, use to build anticipatory user interfaces [182]. Of all the various types of signals one might consider in HCI, this dissertation concerns itself with neuromuscular signals that intentionally convey meaning. In straightforward terms, **gestures** are the non-verbal articulations one uses to communicate.

An individual who thumbs-up another after delivering a spectacular speech gesticulates their positive affirmation as a way of saying "job well done." One who oscillates their index finger in an otherwise closed fist, palm up orientation is perhaps beckoning over an observer, and with a slight change in posture, palm out, he or she is instead ominously warning the other of REƆЯUM[1]. We should note that our definition of gesture is ostensibly consistent with what other HCI researchers report in their work, yet there are subtle differences. Widely cited, Kurtenbach and Hulteen [133] claim that "a gesture is a motion of the body that contains information." However, as we discussed, all signals contain information, but not all are gestures. LaViola [112] less restrictively writes that a "gesture is defined as a dynamic movement," whereas Vatavu and Pentiuc [255] state that "gestures can be defined as a physical movement of hands, arms, face and body with the intent of conveying information and meaning." Although both continue on to discuss non-dynamic postures, these limiting definitions exclude, for example, the peace sign hand gesture. Further, because researchers have shown that eye gaze [56] and foot [221] gestures are a feasible way of interacting with computers, reducing our definition to a subset of one's body would render it incomplete.

It is however correct to say that gestures distinguish themselves from other activities by being deliberate acts of expression, as implied by Vatavu and Pentiuc. Cycling, jogging, loitering, and sleeping are example activities we engage in, but without communicative purpose, and although we may observe the onset of lethargy as one gluttonously shovels a scad of salty stuffed crust pizza slices into their mouth, it is not one's *intention* to communicate their hunger with this action. **Activity recognition**, on the other hand, is a broader area of study that concerns itself not only with mastication but all human activity, of which gestures are a subset. We give special attention to the latter because gestures enable one to directly control computers, whereas interest in other signal and activity types tend towards passive monitoring.

[1]Murder in reverse, a reference to Stephen King's horror novel *The Shining*.

2.2.1 Gestures as Metaphors of Embodied Experience

The opinions described in section are informed by several excellent books [18, 50, 202, 203]. Humans are physical beings who interact with the world through sensory inputs and action outputs, which we correlate via repeated exposure to circumstance. Pressure, temperature, light, sound, orientation, and chemical sensors within our bodies convert stimuli into neurological signals that, while initially processed by different brain regions, are ultimately combined into a single coherent model of reality. Our ability to correlate unique data types into useful information is what enables intelligent action. For example, with repeated exposure to the temporally related taste, smell, color, texture, and nourishing properties of milk, children acquire knowledge of this important resource. As one's linguistic ability evolves over time, one will further come to understand the vibrating pressure patterns that form the word milk, and this acoustic pattern becomes yet another property of the object. With further advancement, he or she will next learn to reproduce the neuromuscular motor commands required to utter milk, and thereafter begin to engage in the polite social interactions necessary to obtain it. Each step along this trajectory is one of correlating data, such as to relate specific motor commands with particular phonemes and understand to what the permutation refers.

An emergent property of this correlating machinery is an ability to generate and understand metaphor, not only in language, but also in action. One example illustrated in Figure 2.2 correlates utterance with vision, where preferred mouth-tongue shape interactions may reflect the visual properties of two abstract images. Another example comes from dance "where the rhythm of movements synaesthetically mimics the auditory rhythm," [204, p. 19] which suggests a cross-activation between auditory input and a motor map[2]. Similarly, gestures themselves are possibly kinematic metaphors of various embodied experiences, grounded entirely by physical interactions with the

[2]In [204], Ramachandran and Hubbard specifically argue there may be a relationship between synaesthesia, creativity, and metaphor.

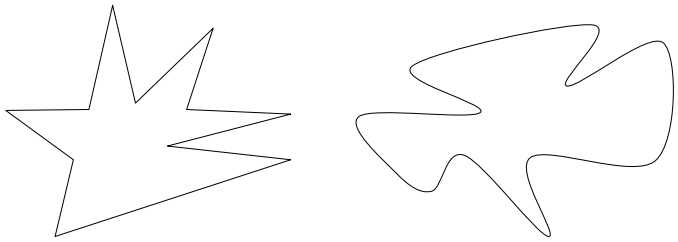


Figure 2.2: Image is based on Figure 1.7 from [204]. Individuals who are told that one shape above is "kiki" and the other is "bouba" are asked to guess to which image each word maps. With 95% agreement, individuals respond that the left and right images are kiki and bouba, respectively. This finding possibly demonstrates a deep metaphorical relationship between vision and utterance, where sharp vocal inflections in kiki mimic sharp lines in the image.

world. Consider, for instance, common gestures like stop, shoo, beckon, a direction cue, or a toddler's arms up plea to be held; although metaphorical, these examples all have a direct correspondence with locomotion. Further, perhaps because we primarily interact with the world through motor actions, we believe that most if not all gesture responses map to kinematic experiences, even when their origins are uncertain, such as those propagated by culture.

Gestures, therefore, further differ from other activities in that they abstract spatial relationships and actions through metaphor. Whereas a subset of activity recognition work focuses on direct action, gesture recognition focuses on indirect, metaphorical action. This view is consistent with our previous definition of gestures in that indirect action is only necessary in a communications context. Now that we have a good sense of what are gestures, let us take a closer look at the various types of gestures.

2.3 Complex Gestures

As there are a variety of activities, there are also a variety of gesture types and taxonomies. Efron [60], Freedman and Hoffman, [74], Ekman and Friesen [61], and McNeill [162] describe four

unique categorization strategies that McNeill suggests represent equivalent actions from differing points of view. To illustrate one, McNeill's scheme comprises four major categories, which are *iconics* that directly resemble what is spoken, *metaphorics* that represent abstract concepts, *deictics* that are pointing actions, and *beats* that convey no discernible meaning other than to provide emphasis. An alternate view known as **Kendon's continuum** (named and arranged by McNeil [120]) supposes a varying degree of speech dependency as follows: *gesticulation* $\rightarrow$ *speech-linked* $\rightarrow$ *pantomime* $\rightarrow$ *emblems* $\rightarrow$ *sign language*. On the left, we have complete dependence, such that gestures cannot possibly be understood in isolation without utterances to provide context, and as we move rightward, we come to complete independence, where full discourse is possible using only nonverbally articulated actions, further demonstrating the power of gestures. Because we focus on gesture- rather than multimodal-based interactions, we loosely tend toward gestures that fall on the right half of this spectrum.

Outside of discursively focused schemes, researchers have also discussed a number of HCI-centric taxonomies; see [6, 75, 119, 186, 199, 266] for a few examples. In the realm of hand gestures, LaViola [112] partitions the space into posture/gesture and simple/complex. Simple postures require full extension or retraction, whereas complex postures are the those that include the in-between. Simple gestures allow for a change in hand position or orientation, but not both, whereas complex gestures have no such restriction. In this work, we focus on complex gestures, although we do not restrict ourselves to just hands. Complex gestures require special attention because of the increased complexity in spotting them amongst other actions that may occur in continuous interactions.

2.4 Kinematic Theory of Rapid Human Movements

A **joint** is a lubricated pivot point where one bone meets another, and **movement** is the product of coordinated muscular actions on a joint [83]. That is, muscles comprising parallel bundles of fibers contract and pull on tendons attached to bones that in turn pivot the system about a joint, which implies that at least two muscle groups are required to orchestrate even simple motion. An **agonist** muscle performs the primary action, accelerating one's limb towards a predetermined target, whereas an **antagonist** muscle counterbalances this action by decelerating the limb. Biceps and triceps are an example agonist-antagonistic pair, where the biceps contract to lift an arm while the triceps relax. Once in place, both muscle groups vary their tensions based on sensory feedback to stabilize and hold the structure steady. At a higher level, muscles working together to produce motion form a **synergy**, which may comprise numerous muscles acting on many joints simultaneously.

To activate a synergy, the motor system must issue well timed commands to various muscle groups while taking into account the present mechanical arrangement of one's body. **Proprioceptors** embedded within skeletal muscles, tendons, and joints, provide sensory information on tension and orientation. Once combined with visual and vestibular sensory input, the motor system compares the combined feedback to a reference signal that represents the desired output. Their difference results in an error signal that a feedback controller uses to issue corrections. Due to delays throughout the nervous system, feedback control is, however, only possible with slow or sequential movements. For other types of actions, the motor system instead uses feed-forward controls, where advanced information and internal models are leveraged to plan and execute movements. Once a feed-forward based action is complete, the motor control system can thereafter use feedback to make further adjustments.

Rapid-aimed movements are one type of feed-forward action in which one quickly and accurately

moves a body part from an initial position to a target position. These movements have received significant attention over several decades because researchers believe that rapid movements are the building blocks of more complex movement. Interestingly, through numerous experiments, researchers have observed that the velocity profile of rapid movements all share common characteristics. For instance, the profiles' shapes are asymmetric and, after reasonable transformations, can be superimposed despite variations in speed, duration, distance, and load. This result holds even under different experimental conditions. The Kinematic Theory of rapid human movement [192] explains these phenomena by supposing that one may describe the synergy of an agonist and antagonist neuromuscular network as a globally linear system. Each network is further a convolution of numerous yet hierarchically-parallel activation functions that converge towards lognormal as a consequence of the central limit theorem. Therefore, the impulse response of a neuromuscular network, whether agonist or antagonist, is mathematically described as:

$$\vec{v}_i(t) = \vec{D}_i \, \Lambda_i(t; t_{0i}, \mu_i, \sigma_i^2) \tag{2.3}$$

$$= \vec{D}_i \left[\frac{1}{\sigma_i \sqrt{2\pi}(t - t_{0_i})} \exp\left\{ -\frac{1}{2\sigma_i^2} \left[\ln(t - t_{0i}) - \mu_i \right]^2 \right\} \right], \tag{2.4}$$

where command i begins at time t_{0i}. Vector $\vec{D}_i$ is the command's amplitude and direction, σ_i is its neuromuscular logresponse time, and μ_i, its logtime delay.

Based on this underlying representation, the Kinematic Theory provides a family of models varying in form and complexity to describe rapid movements [195]. The simplest of these is the Delta-

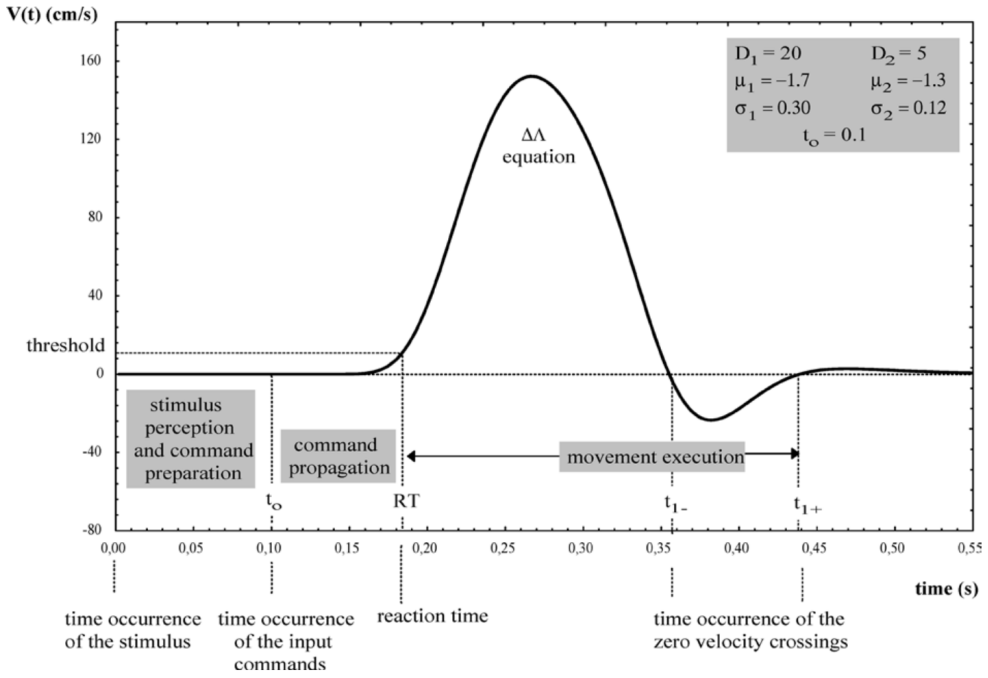


Figure 2.3: Velocity profile of a rapid movement per the ΔΛ model [196]. Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer Nature Biological Cybernetics "A kinematic theory of rapid human movement. Part IV: a formal mathematical proof and new insights," Réjean Plamondon *et al.*, ©2003.

Lognormal ΔΛ model that explains the velocity profile of a single rapid movement:

$$|\vec{v}(t-t_0)| = \left|\vec{D}_1\right| \Lambda_1(t;t_0,\mu_1,\sigma_1^2) - \left|\vec{D}_2\right| \Lambda_2(t;t_0,\mu_2,\sigma_2^2). \tag{2.5}$$

Here we have the synergetic activation of two neuromuscular networks at time t_0 via parallel commands, where a first command $\vec{D}_1$ activates the agonist network and a second, $\vec{D}_2$, the antagonist. Their mathematical arrangement reflects an agonist-antagonistic relationship in that the second command subtracts from the first, mimicking how an antagonist network counterbalances the agonist. Further insights about Equation 2.5 are gleaned through a typical scenario depicted in Figure 2.3. At time $t = 0$ an individual perceives some stimulus that causes him or her to generate a re-

sponse. The associated motor commands are initiated at time $t = t_0$, which propagates through the neuromuscular system as measured by the response time (RT). Once a rapid movement begins, its execution continues until the first or second zero crossing at t_{1-} or t_{1+}, respectively. In this example, we see a reversal where the velocity transitions from positive to negative, which is frequently observed in rapid movements — hooks that occur at the end of pen strokes are one example.

Using these basic ideas, one can build higher levels models that represent more complex motion. This leads to the Sigma-Lognormal model, which we discuss in the next chapter. For now, with a better understanding of human motion in hand, we turn our attention to gesture production variability within and across individuals.

2.5 Variability in Human Motion

A signal embedding human motion will often carry more information than the communication itself, simultaneously and unintentionally revealing properties pertaining to the communicator's age, affective state, and health, as well as other interesting attributes. So although one's communication derives from a particular standard, its form is subject to unknown, unconscious, and uncontrollable influences, resulting in a range of variabilities. This is unsurprising given that the motor cortex works collaboratively with other mental processes, and we should expect from this that a number of factors influence human motion.

We demonstrate the complexity, interconnectedness, and impact of mental processes on human motion by illustrating results from modern cognitive linguistic and neuropsychological research. **Embodied simulation** posits that the same neural tissues we use to perceive and act similarly allow us to simulate perception and action, asserting that reality and simulation both rely on the same "activation of internal encodings of perceptual, motor, and affective ... experiences" [17, p.

142]. In other words, to imagine throwing a basketball involves many of the same neural subsystems as does actually throwing a basketball. One can activate an embodied simulation in a number of ways, including through language. Researchers, for instance, have observed both priming and inhibitory effects on motion-based response times when one has to process language involving movement. Experiments that slow response times ask participants to move in a manner that correlates with simultaneously read text. Embodied simulation proponents argue that this interference occurs because both tasks require access to the same motion processing circuity. On the other hand, response times accelerate when an experiment serializes correlated tasks, a result that provides evidence for motor system priming. Further, functional magnetic resonance imaging (fMRI) studies reveal language involving movement increases cortical motor region activity relative to motion free language, which likewise supports the idea that action and simulation utilize the same neural machinery.

Another interesting example involves **mirror neurons**. These neuronal types discharge both when one performs an action and when one observes another performing the same action [36]. In other words, seeing someone throw a basketball activates some of the same neurons as does actually throwing a basketball. Mirror neurons have received significant attention since their discovery late last century, and mirror neuron systems (MNS) have been implicated in several important cognitive functions, including imitation, learning, understanding intention (theory of mind), empathy, and various social interactions. Giromini *et al.* [84] have even linked mirror neurons with the Rorschach inkblot human movement (M) response. Specifically, for a particular set of inkblots, respondents may indicate through free association that they perceive human action. Researchers place special significance on the M response type because it is thought that to see motion in static images requires imagination and creativity, which draw on personal experiences and possibly embodied simulation. Through fMRI analysis, they observed an increase in both MNS and MT+ activation when participants perceived human motion relative to non-M type responses. MT+ is

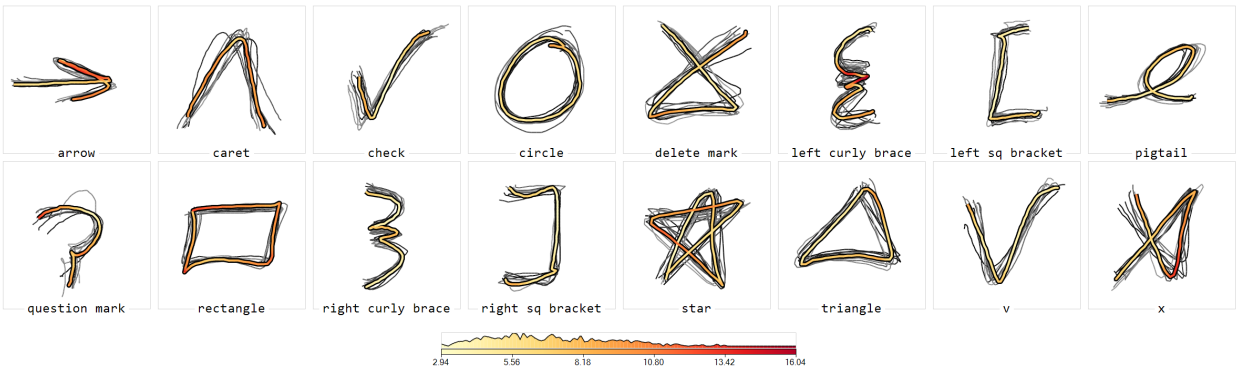


Figure 2.4: Gesture heat map rendered with GHoST [252], illustrating shape errors (ShE) across all gestures drawn quickly by one participant selected from the $1 dataset [267]. ShE [251] measures the average difference in pixels between corresponding points across spatially resampled gestures, in this case, between the dataset centroid and each remaining sample. However, the main purpose of illustration is to show there exists significant variability between gesticulations even from a single individual.

a motion sensitive, visual system region involved in motion perception, which can be invoked by either MNS or true motion activity. One implication of this finding is that even unstructured perceptual stimuli can induce motor activity through mirror neurons.

These examples do not explicitly demonstrate that neural dependencies between language, simulation, and observation directly correspond to gesture production variability, but rather, they inform us that there may exist several systems simultaneously driving motor system action. It would then be reasonable to assume that latency, noise, and asynchronous communications throughout the many cooperating neuronal systems lead to imperfect motion generally, and variable gesture production specifically. However, we all know that the proof of the pudding is in its eating; thus, in Figure 2.4 we illustrate the gesture production variability of one individual who rapidly sketched sixteen gesture types with repetition. Notice that no two instances are identical. We also now take a closer look at some additional HCI-specific results.

Returning to the Kinematic Theory, Martín-Albo *et al.* found that each lognormal parameter follows a different random distribution for handwritten words [161], and by perturbing $\Sigma\Lambda$ model parameters according to the these distributions, they were able to generate realistic synthetic variations. Leiva *et al.* [141] further used the same parameter set to generate realistic synthetic gestures. However, it was found that new parameter distributions were needed to synthesize gestures produced by those with low vision because of differences in gesture production variability between populations [142]. Similarly, Vatavu *et al.* [249] as well as Hernandez-Ortega *et al.* [95] were able to differentiate between children and adult users based on differences in touch interactions. With respect to input devices, we previously observed that the same gesture shapes collected across different device types produced different variabilities [232]. It was also found that gesticulation speed impacts form, where unpracticed rates correlate with increased variability [251].

Emotional state also plays an important role in human motion. Luria *et al.* [150] found that physical and mental stress causes disautomatization of fast and accurate spatial control, which further leads to an increase in handwriting variability over velocity, movement duration, and writing size measures. Similarly, Likforman-Sulem *et al.* [145] were able to demonstrate that stroke variability correlates with different emotional states within a single person. Thus one can predict another's emotional state in gesture stroke data due to stress, anxiety, and depression. With respect to familiarity, Cao *et al.* [30] found that participants over time cut corners when writing well learned forms. But gesture form can also change daily; Liu *et el.* [146] discovered that retraining a custom gesture recognizer between sessions with a new sample that replaces an old one improves accuracy. These results collectively demonstrate that many factors influence motion, not only within one's mind but also when interacting with computers.

2.5.1 Common Types of Variability

Given that no two movements are ever likely to be identical, it is necessary for one to utilize techniques that strip away impertinent information so that the final measure of dissimilarity between query and reference patterns is **invariant** to deviations from a canonical form. In this view, Magrofuoco *et al.* [156] summarize common sources of variability against which a reliable recognizer ought to remain robust, depending on application requirements:

- *Training set invariance*: Established models do not have to be relearned as training samples are added, removed, or modified.
- *Sampling invariance*: Recognition is not affected by an input device's sampling rate, which imparts an additional benefit in that one can use low cost hardware or process input at a low rate when software must attend to unrelated processes.
- *Temporal invariance*: Production duration does not impact recognition.
- *Direction invariance*: A trajectory produced forward or backward is identical, such as a circle drawn clockwise or counterclockwise.
- *Position invariance*: Location of a trajectory within the input device's coordinate system does not impact recognition. A jab gesture produced near or far from a depth camera does not change the fact that one has jabbed.
- *Scale invariance*: Gesture production size is often an unimportant characteristic, where a tiny question mark is no less inquisitive than a large one.
- *Rotation invariance*: A trajectory rotated relative to a query does not impact recognition. For instance, an axis-aligned square is identical to diamond, at least in shape.

- *Reflection invariance*: Allows for mirroring over a hyperplane through the input coordinate space. One practical example of the need for reflection invariance occurs when one trains their system with data collected on a first device that has an axis inversion relative to a second system.

In certain applications, however, invariance can be harmful, where with trivial transformations and under certain measures, one gesture can be made to resemble another. For example, direction and rotation invariance can result in a system that is unable to distinguish left from right square brackets. One can resolve such ambiguities in a post processing step, or (as a feature of the recognizer) allow for variations that separate such gesture classes when appropriate.

Given the extent of human motion variability under even mundane conditions, one can appreciate the difficulty in training a gesture recognition system to recall a wide range of patterns from only one or two training samples. Yet, this is exactly what user interface customization requires, which we discuss next.

2.6 The Virtues of Gesture Customization

Nacenta *et al.* [169] describe four gesture set types: stock gestures, pre-designed gestures, user-elicited gestures[3], and user-defined gestures. Generic system-wide available actions such as pinch, rotate, and swipe are stock gestures that are widely available to designers for use in their own software. Stock options are likely limited and therefore least expressive, as their shoehorned inclusion into a user interface may result in rather ambiguous associations, *e.g.*, pinch to open a jogging playlist. Alternatively, designers who leverage their expertise and domain knowledge can

[3]Nacenta *et al.* actually filed user-elicited gestures under pre-designed gestures, but for reasons soon explained, we keep them separate.

pre-design gesture sets that are in their view meaningful, memorable, and well separated in a recognizer's feature space. Or, when time permits and sufficient resources are available, designers may instead choose to conduct an elicitation study so as to determine what is an appropriate gesture set for their interface. Finally, as it pertains to customization, designers can allow users to define their own gestures. Herein, we are concerned primarily with this latter type. We believe that support for gesture customization in user interface design inherently possesses functional, practical, and cognitive virtue.

As a functional virtue: The need for customization becomes apparent in designs where pre-defined gestures, user-elicited gestures, and stock gestures are inadequate or unworkable. For instance, one may envision a supernatural game where players create their own wand spells using an HTC Vive controller, or a system where meaningful shortcuts are required for an unspecified number of referents. Custom gestures are further useful in security and authentication, such as to increase password entropy and complicate shoulder surfing attacks [175]. As a final example, those with motor impairments are sometimes unable to interact with a given device using typical able-bodied articulations [7]. Customization can overcome this issue by allowing impaired individuals to define gesture sets that accommodate their impairment.

As a practical virtue: Once a designer decides that stock gestures are inadequate, he or she may consider using pre-designed gestures, yet designers often get it wrong [167]. We are often unsure of how users want to interact with our software; or what seems appropriate to us as experts turns out to be awkward and unfamiliar to others. However, we can overcome this issue with iterative design and elicitation studies. Norton *et al.* [176], for instance, conducted a Wizard-of-Oz study to explore full body interaction in a parkour game environment. Wobbrock *et al.* [266] conducted an elicitation study to learn how non-technical users gesture on interactive tabletops. Chan *et al.* [37] performed a similar study for single-hand microgestures (SHMG). Such investigations yield workable gesture sets and recommendations for user interface design within their respective

domains that, while being quite useful, have two limitations. First, established findings are specific to their domain, *e.g.*, an SHMG consensus set is likely to be of little value to one who is designing a Vive game. In other words, if prior knowledge is unavailable or without direct application to one's interface, he or she will still need to conduct their own study, which, to the second point, can be time and cost prohibitive. Finding sufficient time, participants, and capital to run a user study is sometimes no small feat. These reasons make customization an attractive *practical* alternative, especially to the indie developer, prototyper, hobbyist, and student.

As a cognitive virtue: It should be understood that gestures are personal, even artful. World-renowned magician Raymond Teller of the Penn & Teller Las Vegas headliner act is well known for his silence while performing, relying on his gestures in combination with Penn Jillette's story-telling to work magic. Being a master of the art, Teller is one of the few magicians who ever successfully battled another in court, not because the defendant exposed Penn & Teller's "Shadows" secret (as tricks are not protected under US law), but because the defendant stole his expressions, his pantomiming, his art[4]. This example illustrates just how personal gestures can be. When creating custom gesture sets, users tend to prioritize the intuitive, natural, and obvious first; the simple and easy second; and the familiar third [177]. But the result is still something unique; that is, we each have our own ideas about which actions ought to map to which functions based on internal conceptualizations, associations, and perceptions. This may be why we never see perfect agreement for all tasks in elicitation studies, and why custom gesture sets require less concentration and are easier to remember [169].

These virtues compel us to focus on customization and consequently techniques that work with very little training data. Because if we wish to support gesture customization, we must also ensure we do not burden the user [143]. Yet how we choose to go about designing such a system is of

[4] https://www.scribd.com/doc/213767662/Teller-v-Dodge-March-2014

equal importance, which leads us next to $-family principles.

2.7 The Virtue of Straightforwardness

HCI researchers often engage in rapid prototyping to iteratively design, evaluate, and refine user interfaces. Some researchers further incorporate custom gestures into their software so as to facilitate natural and fluid interactions. Since HCI researchers vary in skill, it is common to find people in the field who are familiar with neither machine learning nor the advanced mathematics employed by pattern matching techniques that enable gesture recognition. $-family recognizers[5] serve these individuals by offering straightforward solutions to complex pattern matching problems, so that researchers can incorporate custom gesture recognition into their interfaces with comparatively little effort. Wobbrock *et al.* introduced the first $-family recognizer, $1 [267], after which a long and varied series of work followed. It is because of their reliability, relatability, simplicity, and adherence to the fundamental principles outlined below that $-family recognizers have been widely adopted by the community[6].

We too are motivated by the $-family design philosophy in order to provide a robust solution that is accessible to a large and varied talent pool, ranging in background from developer to designer, and in skill from neophyte to expert. Tenets of the philosophy stand on:

1. *Independence*: One can implement a proposed technique from scratch without requiring external library support, thereby enabling gesture recognition on new platforms as they obtain.

2. *Foundational Mathematics*: Computations involve only straightforward algebra, geometry,

[5]For simplicity, we refer to the core recognizers [8, 9, 143, 248, 250, 253, 267] and those inspired by as the $-family collectively.

[6]Impact of the $-family is discussed at http://depts.washington.edu/acelab/proj/dollar/impact.html

and statistics that most learn early in their academic careers.

3. *Relatable Representations*: How a method internally represents a class model is straightforward, utilizing constructs that are easy to understand and visualize.

4. *Basic Algorithms*: Techniques employ algorithms that can be understood by those who have completed their first computer science course or have equivalent experience.

5. *Competitive Accuracy*: Recognizers achieve high accuracy with limited training data so as to avoid burdening users and not push integrators toward more complex solutions.

6. *Customizability*: New gesture classes and examples can be added to the system at runtime without significant effort or delay.

These criteria collectively ensure that techniques are fast to implement, require minimal code, are easy to debug, and, as a result, are accessible to a range of skilled individuals. Some example rapid prototyping algorithms that have been deemed appropriate include uniform and stochastic spatial resampling; optimizations using a golden-section search; rotation, scale, and translation transformations; point cloud matching; and permutations with Heap Permute and Fisher-Yates. In this work, we adhere to these principles where possible, but the move into high dimensional, continuous, and noisy data requires greater sophistication. The techniques we employ herein can still be understood by upper level undergraduate computer science students who do not possess specialized statistical, mathematical, or machine learning knowledge. Each technique in isolation is rather straightforward, but in aggregate the pipeline is a sizable engineering effort relative to $-family-like recognizers. For these reasons, we refer to our work as *practical* rather than rapid or advanced.

2.7.1 Why Not Just Use a Library?

On occasion, we hear that one can simply leverage an unspecified library that solves the custom gesture recognition problem, implicitly dismissing any inherent merit found in simplicity. This sentiment supposes that complex solutions need not be accessible, just available. But therein lies at least one problem—many researchers do not release their software. Still, even if every solution came with a complete specification via source code, pseudocode, or both, there still remains a variety of issues, both technological and psychological, that in our opinion support straightforward design. We describe the resolution to each issue as a freedom:

1. *Freedom from license restrictions*: Often researchers provide reference code with liberal licensing, but more restrictive options may be incompatible with proprietary licenses and allow neither commercialization nor derivative work. These restrictions may prohibit some from using a particular library.

2. *Freedom from uncertainty*: When a system produces suboptimal results, then one may be uncertain whether there is an issue with the library itself or their use of the library. Having full knowledge of a system's form and function removes this uncertainty and allows designers to manage associated risk.

3. *Freedom from maintainer dependence*: Similarly, when a library produces erroneous results, one who understands how the underlying algorithms work can take on issues directly, rather than wait for maintainers to address problems as they arise. Straightforwardness facilitates this kind of independence.

4. *Freedom from a fixed API*: When a designer or researcher requires information not provided by an API, he or she is more likely to write software from scratch to his or her specification, which, again, algorithmic straightforwardness facilitates.

5. *Freedom to derive custom solutions*: In a similar manner, inspired practitioners will find they are better able to incorporate novel measurements, context, or application specific criteria into a recognition approach that is easy to implement and modify.

It is clear, however, there are numerous advantages one will find in using well tested libraries, and we do not claim their use is improper, of poor taste, or unnecessary. Our goal here is only to illustrate that there exist distinct advantages afforded to one who is able to implement their own functionality, and that straightforwardness is a key element of each advantage. One last concern remains though. Even if a solution is simple, it is often useless if one cannot maintain interactive rates, which is why we turn now to algorithmic complexity and speed.

2.8 The Virtue of Speed

Since $1's inception, researchers have put a concerted effort into optimizing recognizer performance for three important reasons:

1. *Real-time Performance*: User interface design often requires low latency feedback in order to maintain fluid interaction. For instance, it has been shown that users perceive, plan, and react to millisecond delays [86]. And although numerous applications do not require real-time performance, we still prefer responsive feedback. Further, processing queries should not delay other ongoing operations, as software does much more than just analyze input—it must share hardware resources with other system processes [238].

2. *Low-Resource Devices*: Mobile, wearable, and embedded devices employ low-resource computing technology to preserve power and lower cost. To enable gesture recognition on such devices, we require highly efficient techniques [253].

3. *Quantity*: In general, recognizer accuracy improves with training set size, whereas responsiveness declines under an increased workload. Intuitively, algorithmically and computationally efficient techniques are able to process more templates per unit time without compromising runtime performance requirements [238].

For these reasons, we are conscientious of performance in our design of all Dollar General techniques.

2.9 Summary

We have now laid a foundation and tone for the remainder of this dissertation. Further discussions are framed in the context of capturing and interpreting complex human motions with respect to customization, simplicity, and speed. In other words, the unique intersection where all four concerns meet is were we focus our attention, beginning with a more detailed look at related worked.

CHAPTER 3: RELATED WORK

There's lightning striking all over the world
There's lightning striking all over the world

I am a human being, capable of doing
beautiful things . . . Run!

AWOLNATION
Run (Beautiful Things)

Even before the arrival of digital computers, researchers worldwide advanced the art of pattern recognition through the development of computational and statistical methods. This is because machine learning to a large extent involves data representation and information interpretation techniques; and while truly impressive insights continuously improve our ability to recognize complex patterns—radial basis functions in support vector machines—the insights we appreciate most herein are those that achieve high accuracy in the simplest way possible, which in and of itself is hardly ever easily acquired knowledge. To better appreciate our motivation, it is necessary to explore both the complex and straightforward in gesture recognition and that which relates to but goes beyond gesture recognition. This exploration therefore requires that we cover data acquisition, feature extraction, measurement, segmentation, and synthesis.

As one reads this chapter, one should keep in mind the following questions. First, does a given technique enable seamless customization? If so, it achieves high accuracy with very little training data. The technique is online and able to utilize new training data, without delay, so that new gesture classes are immediately available and recognizer accuracy improves on average. Second, can one implement the technique without significant effort? If so, one can use a method directly without requiring external library support. One will not have to acquire machine learning knowledge or

mathematical expertise beyond that which can be completely contained in a typical conference paper describing the technique. These are the questions and responses that motivate each component in The Dollar General continuous gesture recognition pipeline, and is the context of our related work.

3.1 Working With Noisy Data

Humans interact with computers through input devices whose signals are perturbed by thermal noise, interference, precision errors, and other types of corruption-causing phenomena that affect performance [187, 197]. An augmented reality interface designed for selection and manipulation or that uses a pointing device may lack precision due to noise-induced jitter, and a virtual reality, full-body gesture interface may lack discriminatory power due to inflated feature variance. For these and other reasons, practitioners often use low-pass filters to improve signal quality before evaluating embedded content. Such filters attenuate high frequency noise while preserving low band information, as is appropriate given that humans interact through low frequency motion [276]. However, noise is not one's only concern, as variability in motion similarly impacts signal quality, and over aggressive filtering may lead to lost information.

More specifically, in calibrating a low-pass filter, one must consider the impact of both precision and lag on performance [35, 187]. When a signal is corrupted by noise, repeated measures of a motionless input device yields a varying response that is realized as an imprecise, jittery pointer. Aggressive filtering can attenuate noise and improve precision, but doing so introduces latency, which causes one's response to lag behind its true position. Due to the detrimental effect of noise and latency on throughput, practitioners in one context must strike a balance between speed and accuracy while simultaneously considering their unique application requirements. In another context, such as gesture recognition, signal shape and smoothness may be more important than latency,

and time shifted response data is perfectly acceptable.

In a lab setting, one can achieve acceptable performance by using trial and error calibration. However, additional considerations influence parameter selection when conducting user studies or deploying software into unknown environments. For example, one may deploy their software across multiple platforms via a transreality system where one must support a variety of input devices [159]. Noise levels can vary between input devices as well as individuals based on sensor, software, and environmental factors. A computer vision system may have difficulty tracking certain skin tones or clothing colors under various lighting conditions and backgrounds. An individual with a degenerative disease such as Parkinson's may inadvertently add additional noise to the input signal. Unlike noise, latency depends on speed, where fast moving individuals will experience lag different from those who move slowly. These factors all influence how one should tune their filter.

3.1.1 Low Pass Filters in the HCI Community

Low-pass filters are used throughout a variety of domains, including HCI, especially for tracking and smoothing user input. For example, Xiao *et al.* [270] focused on jitter reduction to improve motion estimates; Dabra *et al.* [47] smoothed Kinect sensor motion input; and Feit *et al.* [69] used filtering to track eye movement. Filtering has been used in augmented reality system by Liang *et al.* [144] and Zhu *et al.* [280], as well as in virtual reality [262]. These selected examples are only a few among many.

In all cases, one has to make some kind of decision about how to calibrate their filter. Options fall into three broad categories: manual and interactive, machine learning, or adaptive calibration. Manual tuning is common, where one uses trial and error to fine tune parameters according to preference and taste. This approach can be effective and easy for low noise levels, but may require

expert knowledge or prior experience depending on task and filter complexity. One practical example involves the 1€ filter [35] that comprises two parameters, enabling one to balance jitter and latency, for which the authors propose a manual calibration procedure. The filter is very effective, but can be enhanced with an automatic calibration procedure.

3.1.2 Filter Tuning with Machine Learning

Researchers have explored a variety of machine learning techniques for parameter selection. Saha *et al.* [217] and Borah *et al.* [25] used particle swarm optimization. Horvath *et al.* [101] utilized a genetic algorithm. Russo [216] used a histogram of edge gradients for image denoising. Harshcer *et al.* [91] used a gradient descent method to optimize physical screw turns for their filter (a non-HCI example). Chen *et al.* [41] combined a median filter with an automated parameter tuning technique.

Grid searches (or parameter sweeps) are especially common. In this approach, each parameter's range is divided into a set of regularly spaced intervals that when taken collectively form a grid pattern. Thereafter, one iteratively walks over the grid in order to exhaustively evaluate each parameter combination. LaViola [136], for example, used a grid search to tune the double exponential moving average and Kalman filters to minimize the root mean squared error for position and orientation data.

An issue with machine learning based approaches is that one requires both training and ground truth data to evaluate their filter. The data may also require cleaning or other preparation work, a process that typically requires domain knowledge. These issues make it difficult for one to support custom calibration because of the large amounts of data one needs to collect in order to cover all scenarios.

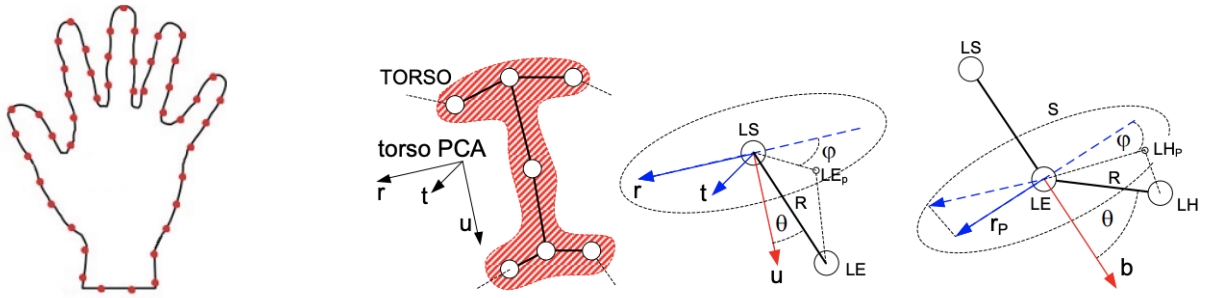
3.1.3 Adaptive Filters

Another popular tuning approach involves the use of adaptive filters. Given a first primary input signal and a second reference input signal, an adaptive filter adjusts parameter weights until differences measured by an error function are minimized. Research in adaptive filtering primarily focuses on specialized filters [3, 19, 127, 226] and optimizations for fast convergence [104, 168, 241], but such filters have also found their way into HCI applications. For instance, Bulling *et al.* [27] applied a filter to remove unwanted noise found during eye tracking with an EOG device; Lee et al. [140] devised an adaptive filter for an ECG device; and Kaluri *et al.* [117] used an adaptive filter in the context of gesture recognition.

3.1.4 Pitch Pipe: A New Approach

In Chapter 4, we propose a new filter calibration technique that is online, passive, and automatic. As such, we do not require interactive feedback, training data, or a second reference signal. Pitch Pipe monitors an input device signal and given a low-pass filter, determines near optimal parameters. Pitch Pipe itself is not a low-pass filter, but rather, it is a wrapper method that uses synthetic data based on characteristics of the input signal—namely, the amplitude of low band frequencies and noise present in the highest band—to optimize the parameters.

After smoothing a signal, one may transform their data into a form amenable to gesture recognition. The nature of this transformation depends in part on the underlying machine learning technique, and even then, there are often many suitable options. In the next section, we explore some of those options as well as issues involved in data representation.



(a) Kulshreshth *et al.* [132] ©2013 IEEE. Image included with permission.

(b) Raptis *et al.* [205] Copyright ©2011 ACM, Inc. Included here by permission.

Figure 3.1: Examples data representation options for skeleton data. In (a), Kulshreshth *et al.* [132] form a 1D signal comprising point distances from a hand's centroid along its contours that they convert to Fourier descriptors. In (b), Raptis *et al.* [205] determine a skeleton's orientation from principle component analysis on torso joints and then build a representation of joints from spherical coordinates. LS, LE, and LH denote the left shoulder, elbow, and hand, respectively.

3.2 Data Representations For Gesture Recognition

Before one can select a pattern matching algorithm, one must first decide how to represent human motion. For instance, one for may work with raw data directly, reduce their data, transform their data into a set of features, or a combination thereof. Example data representations are shown in Figure 3.1. To recognize hand number poses for menu selection, Kulshreshth *et al.* [132] use Fourier descriptors. Specifically, from RGB+D image data, they extract and resample the hand's contour. They then calculate each point's distance from the hand's centroid to form a 1D signal that they thereafter convert into a Fourier descriptor representation using DFTs. Since Fourier descriptors are scale, position, and orientation invariant, they simply use Euclidean distance nearest neighbor pattern matching to classify each hand pose. The key here is that because they select a representation invariant to pose differences, they are able to use a simple distance measure for gesture classification.

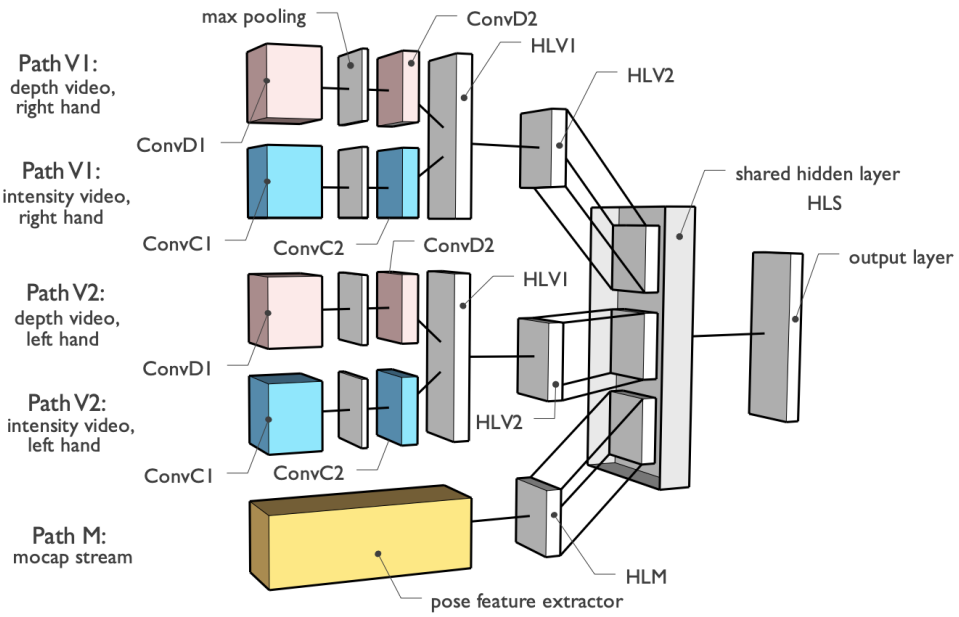


Figure 3.2: Architecture proposed by Neverova *et al.* [173]. To prepare for classification in a deep neural network architecture, depth and intensity data for right and left hand input first undergoes a series of convolution and max-pooling stages before being merged, where thereafter features are extracted and the output calculated. Further, first layers (I) performs 3D convolutions, while the second layers (II) perform 2D convolutions. This approach requires a deep understanding domain data and how it is manipulated throughout all stages. Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer Nature Springer eBook "Multi-scale Deep Learning for Gesture Detection and Localization," Natalia Neverova, Christian Wolf, Graham W. Taylor, et al. Copyright ©2015.

To represent skeleton pose data in dance move motion patterns, Raptis *et al.* [205] first perform principle components analysis on torso joints to find orientation data. The first principle component is said to align with the torso's spine and its second component aligns with an individual's shoulder to shoulder cross section. The cross product of these two vectors then defines an individual's forward facing direction. First order and second order joints are thereafter converted into spherical coordinates, where first order joints are connected to the torso, and second order joints are connected to first order joints. This representation not only reduces data size, but is also invariant to scale and position differences.

Another common approach researchers use is to represent skeleton poses as a set of pairwise measures between all joints. For example, Ellis *et al.* [63] use Euclidean distance to convert 15 pairwise joint differences into 225 scalar values. Zhao *et al.* [279] use a similar scheme except they also normalize each value by its minimum graph distance between joints (number of bones traversed from one joint to the other).

Data transformations may also be learned and selected for by deep learning techniques. Although we discuss deep learning later in this chapter, the hand crafted architecture [173] presented in Figure 3.2 is designed to fuse left and right hand data from depth and intensity image data for gesture recognition. Their system works by convolving and down sampling input throughout various stages until the machine is able to extract features that it can use for classification.

The purpose of this discussion is to highlight that there are numerous way in which one can prepare and represent human motion data collected from various input device types. Representations are often domain specific, such as the skeleton specific examples we explored above. Within our own work, we try to avoid modality dependent data presentations by using motion direction vectors. In the remainder of this section, we take a closer look at features and feature engineering to help build our case for why basic intuitive representations are preferable.

3.2.1 Features

Many gesture recognition techniques first transform trajectory data into features that mathematical models use to infer class membership, where each **feature** is a measurement on data. There are a variety of properties one can measure with respect to human motion, including geometric and temporal attributes. Example canonical features designed for 2D mouse gestures by Rubine [215] are illustrated in Figure 3.1. This initial set of twelve featured are surprisingly useful, have intuitive interpretations, and have been generalized by others for 3D gesture recognition [40, 99, 236]. Later

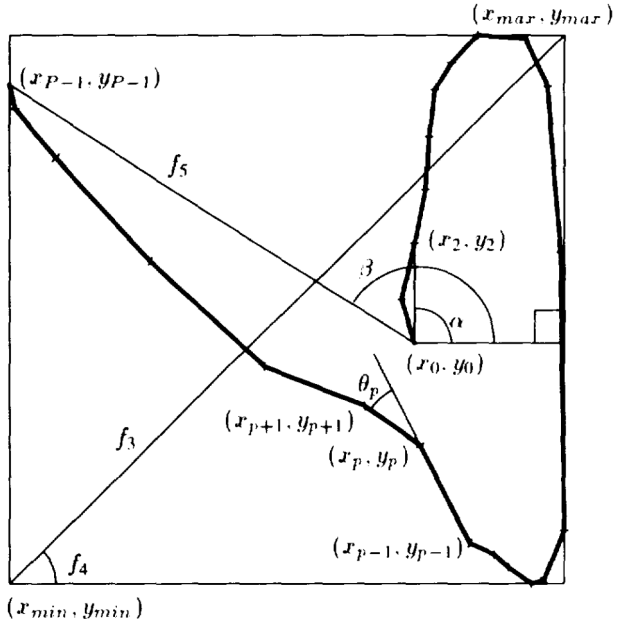


Figure 3.3: Classic set of extracted features for 2D mouse gesture recognition [215]. Copyright ©1991 ACM, Inc. Included here by permission.

Table 3.1: Classic set of features for 2D mouse gesture recognition [215].

Description	**Equation**	**Definitions**
Initial angle	$f_1 = \cos\alpha = (x_2 - x_0)/d$	$d = \sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$
	$f_2 = \sin\alpha = (y_2 - y_0)/d$	
Bounding box length	$f_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}$	
Bounding box angle	$f_4 = \arctan\frac{y_{max} - y_{min}}{x_{max} - x_{min}}$	
First to last point distance	$f_5 = \sqrt{(x_{p-1} - x_0)^2 + (y_{p-1} - y_0)^2}$	
First to last angle	$f_6 = \cos\beta = (x_{P-1} - x_0)/f_5$	
	$f_7 = \sin\beta = (y_{P-1]} - y_0)/f_5$	
Path length	$f_8 = \sum_{p=0}^{P-2}\sqrt{\Delta x_p^2 + \Delta y_p^2}$	$\Delta x_p = x_{p+1} - x_p$ and $\Delta y_p = y_{p+1} - y_p$
Total angle traversed	$f_9 = \sum_{p=1}^{P-2}\theta_p$	$\theta_p = \arctan\frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1}\Delta y_p}{\Delta x_p \Delta x_{p-1} + \Delta y_p \Delta y_{p-1}}$
Total absolute angle	$f_{10} = \sum_{p=1}^{P-2} \mid \theta_p \mid$	
Squared angle (sharpness)	$f_{11} = \sum_{p=1}^{P-2}\theta_p^2$	
Maximum speed	$f_{12} = \max_{p=0}^{P-2}\frac{\Delta x_p^2 + \Delta y_p^2}{\Delta t_p^2}$	$\Delta t_p = t_{p+1} - t_p$
Duration	$f_{13} = t_{P-1} - t_0$	

work continued to explore a variety of diverse and innovate feature types for not only gestures but also for shapes and diagrams [22, 51, 184], some of which are similarly straightforward; though some are less trivial. For instance, certain features require convex hull calculations, cusp detection, or use histograms, and still other features, like Hu moments, [103] lack natural human relatable interpretations.

3.2.1.1 Feature Distributions

Due to variability in human motion, every gesticulation is unique across all meaningful measures. In other words, features are random variables, where each follows their own unique distribution, some being normal, lognormal, or exponential, some being nonparametric, and others being none of the above. Practitioners frequently assume normality, but in fact many distributions are not. This implies that individuals do not always exploit the full potential of a given feature, which can in certain circumstances be overcome with a transformation.

Another issue is that features vary in scale, range, and units. Whereas one feature may measure millimeters, another may use radians. To ensure that no one feature dominates another or leads to numerical instability, one can **normalize** each distribution so as to unify location, scale, or both [102]. Two common normalization approaches are min-max and z-score. The former follows:

$$z_i = \frac{x_i - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} (b - a) + a, \tag{3.1}$$

where $\mathbf{z}$ is the transformed distribution that has been translated and scaled to $z \in [a, b]$. When one transforms a distribution to the unit interval then the boundary sub-expression containing a and b disappears (*i.e.* $a, b = 0$).

Z-score normalization can be written as:

$$z_i = \frac{x_i - \mu}{\sigma}, \tag{3.2}$$

where μ and σ are the feature's mean and standard deviations, respectfully. Other normalization techniques include decimal scaling, median, median absolute deviation, double sigmoid, tanh-estimators, and biweight estimators [111], though they are less common.

One may also use domain knowledge to normalize their data. For instance, when working skeleton data, we previously scaled all bones to unit length, working from the hip joint outwards [236], whereas Ellis *et al*. scaled skeleton poses by the population's spine length variance [63]. To complicate matters further, however, blind normalization can backfire. For instance, in dynamic time warping, it is common practice to z-score normalize time series data before measuring its dissimilarity to a reference pattern [201]. If a signal or component within a multidimensional signal contains no information, such as a shoulder joint that does not move during gesticulation, then normalization scales noise to the same range as those components that contain relevant information, which can result in higher dissimilarity scores.

3.2.1.2 Feature Engineering

Good features maximize the statistical distance between classes, which correlates with low intra-class variability and high inter-class variability. In other words, good features separate gesture classes. Toward this end, practitioners often engineer novel features using domain knowledge, such as for circuit diagrams or modeling sketch recognition. So while there exists general purpose features suitable for a wide variety of applications, *e.g.*, HBF49 [51], acceptable features in one domain may perform poorly in another. For this reason, expert knowledge coupled with machine

learning intuitions go a long way toward successful deployment. Unfortunately, this knowledge is not immediately available to all.

To summarize feature use in machine learning, one must engage in one or more of the following:

1. Engineer novel features using domain knowledge to facilitate gesture class separation, which involves trial and error and knowledge of feature analysis to effectively execute.
2. Select those features that are most applicable to one's situation, which may involve one or more feature selection strategies.
3. Determine each feature's distribution, e.g., normal, lognormal, double exponential, etc.
4. Find appropriate statistical distribution transformations if required.
5. Estimate population parameters in order to normalize the features.

We believe most of these steps are unknown to those without prior machine learning experience, which includes a significant portion of researchers designing user interfaces. To meet our Dollar General objectives, we use intuitive data representations and, in some cases, intuitive features that are nonparametric.

3.3 Gesture Recognition

In this section, we discuss common machine learning approaches, providing only enough information so that the reader will have a high level understanding of each technique. For an in depth discussion of these techniques applied to gesture recognition, we refer the reader to Carcangiu's recent dissertation [34, Ch 2]. In the next section, we cover $-family techniques, taking a close

look to how they handle invariance and measurement. Part one frames the second by enabling one to compare and contrast advanced techniques with their more straightforward counterparts.

3.3.1 Naïve Bayes

One of the easiest to understand statistical approaches in machine learning is the Naïve Bayes classifier, which as its moniker suggests, is an application of Bayes' theorem. Given a data representation (*i.e.*, set of features) and sufficiently large dataset, one can estimate each feature's probability density function per class. If one further assumes that all features are independent, then the probability that a given query belongs to a particular class is the joint probability of features under the class's distributions. Given multiple classes, we infer that the query belongs to whichever class maximizes its joint probability. Further, if the likelihood that gestures are non-uniform across the gesture classes, one can adjust the joint probability by its prior probability.

Written out mathematically, the probability that a given sample $\mathbf{x}$ belongs to class $g_i \in \mathbb{G}$ is given by:

$$P(g_i \mid \mathbf{x}) \propto P(g_i) \prod_{j=1}^{|\mathbf{x}|} P(x_j \mid g_i). \tag{3.3}$$

In practice, we usually assume that the prior class probability is uniform, and so we drop $P(g_i)$. It is also common to assume that features are drawn from a normal population, so one only needs to compute each feature's mean and variance, per class.

Despite its simplicity, Naïve Bayes can perform well, depending on the situation. For instance, Cheema *et al.* [38] were able to demonstrate high user dependent recognition accuracy over Wii remote accelerometer data that was inline with SVM and LDA accuracy, which outperformed C4.5

Decisions trees [200] and AdaBoost [76]. However, in user indepedent scenarios, Naïve Bayes fell far behind.

3.3.2 Linear Discriminant Analysis

The Naïve Bayes independence assumption is limiting when features are correlated. Overlaps in the joint probability distribution space can sometimes be resolved when one takes such correlations into account, which is one reason for using linear discriminant analysis (LDA). If we assume that features are normally distributed or can be transformed into a normal distribution, we model the feature probability density function as follows:

$$P(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}, \tag{3.4}$$

where x is the mean, σ is the standard deviation, $-\infty < x < \infty$, and $0 < \sigma$. However, when working with multiple normally distributed features, we can exploit correlations via the multivariate normal distributions:

$$P(\boldsymbol{X} = \mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}, \tag{3.5}$$

where $\mathbf{x}$ is a multidimensional point comprising p components, $\boldsymbol{\mu}$ is the distribution's mean, and $\boldsymbol{\Sigma}$ is its covariance matrix.

If we next assume homoscedasticity, a series of simplifications leads to the following straightforward classification rule:

$$\text{gesture} = \underset{g_i \in \mathbb{G}}{\operatorname{argmax}} \left(\boldsymbol{w_i} \cdot \boldsymbol{x} + c_i \right), \tag{3.6}$$

where $\boldsymbol{w}_i$ are the feature weights and c_i is a constant for class i, defined as follows:

$$\boldsymbol{w}_i = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i \tag{3.7}$$

$$c_i = -\frac{1}{2}\boldsymbol{w_i} \cdot \boldsymbol{\mu}_i. \tag{3.8}$$

Problems arise when the covariance matrix is singular since one cannot invert singular matrices. To overcome this issue, practitioners must be conscientious of their feature set and relationship between features so that they can hand pick those that are least likely to cause issues. Or one may engage in an automatic feature selection process that selects the most discriminant features that do not lead to singularities.

Perhaps the most famous example of LDA in gesture-based user interface design is due to Rubine [215]. He engineered the feature set previously discussed in Section 3.2.1 (shown in Figure 3.1) to use with LDA and was able to obtain high recognition accuracy. Further, he was an early adopter of customization, enabling researchers and practioners to design gesture sets by example rather than through specification using a high level description language. This important work has inspired and continues to inspire a large body of work in gesture recognition research. Yet, feature engineering issues and linear algebra requirements (namely the need to support matrix inversion) limit the technique's accessibility.

3.3.3 Support Vector Machines

The support vector machine (SVM) is a linear binary classifier, similar to LDA and perceptrons. However, SVM's learning algorithm is what differentiates this method from other methods. Specifically, SVM learns which hyperplane best separates two classes, where the hyperplane is defined

as:

$$\boldsymbol{w}\boldsymbol{x} + b = 0. \tag{3.9}$$

The partition into which a point falls determines its class membership—whether the point is above or below the hyperplane.

It is common practice to map data into high dimensional spaces when they are inseparable in their lower dimensional space. For instance, one cannot separate the unit interval from other numbers with a single decision boundary on a number line. However, if one maps each data point into a 2D space using $\phi(x) = (x, x^2)$, one can easily find a line that separates the interval from other values. When working with a transformation function, we rewrite the hyperplane as:

$$\boldsymbol{w}\phi(\boldsymbol{x}) + b = 0. \tag{3.10}$$

One issue with this formulation is that certain transformations map data into spaces that are too large or computationally expensive to work with directly. Fortunately, there exist techniques that circumvent this issue. First, note that the weight vector $\boldsymbol{w}$ can be rewritten as

$$\boldsymbol{w} = \sum \alpha_i y_i \phi(\boldsymbol{x}_i), \tag{3.11}$$

where α_i is the Largrange multiplier for instance i, and $y_i \in \{-1, 1\}$ is its class label (see [274] for more detail). And so the decision function can be written as:

$$\hat{y} = \text{sign}\left[\sum \alpha_i y_i \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{q}) - b\right]. \tag{3.12}$$

Now, one may use the so-called kernel trick in place of the dot product to evaluate the similarity

of training sample $\boldsymbol{x}_i$ to query $\boldsymbol{q}$.

$$\hat{y} = \text{sign}\left[\sum \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{q}) - b\right]. \tag{3.13}$$

An interesting thing happens here. The kernel function K replaces the high dimensional inner product without first transforming data between spaces, yet it produces the same result. This is highly beneficial given that certain transformations are intractable. To illustrate, the radial basis function (RBF) is a popular transformation that projects vector into an infinite dimensional[1] space, but one whose inner product is trivial to calculate in its original space using:

$$K(\boldsymbol{x}_i, \boldsymbol{q}) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{q}\|^2}{2\sigma^2}\right). \tag{3.14}$$

To train an SVM, one may use quadratic optimization to find the hyperplane weight and bias parameters. Further, there are numerous kernels available to practitioners with which one may wish to become familiar—to ensure he or she is selecting the best transformation for their data. Let us now discuss a few SVM gesture recognition specific applications.

Patsadu *et al.* [185] used a polynomial kernel to train an SVM to recognize three gestures with 99.75% accuracy, which significantly outperform Naïve Bayes and a decision tree, but not a back prorogation neural network. They used pose data as input. Marin *et al.* [157] engaged in hand feature engineering for Leap Motion-based hand gesture recognition and designed features based on finger orientation, distance, and elevation relative to the palm's plane. They use a multiclass one-vs-one RBF kernel SVM to classify 10 gestures with 80.86% accuracy. Marin *et al.* further combined Leap Motion data with Kinect input (and new features) to achieve 91.28% accuracy. Dardas *et al.* [48] work with image data. They first extract scale invariant feature transform (SIFT)

[1] See http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf

features that are then codified via clustering and finally combined into a histogram. Their system uses these histograms as training and query data points within their SVM framework, achieving 96.23% accuracy over six gesture classes.

3.3.4 Hidden Markov Models

The principle idea of a Hidden Markov Model (HMM) is that a signal arises from an underlying process that one cannot directly observe. The process is hidden. However, we *can* observe its output, the signal. With enough training data and some intuition, we can further infer a model that describes the process. Once we know the model, we can then use it to estimate the probability that a new query signal originates from the same underlying process.

HMMs are popular in gesture recognition research because, in our view, they provide us with a natural map from internalized models to observable motion, as well as a method of evaluation. Recall from Chapter 2 that when one decides to gesticulate, he or she forms an action plan based on an internal (hidden) model of the gesture. Even though motion is continuous, we can describe the internal model as comprising N discrete states $S = \{s_i \mid 1 \leq i \leq N\}$, organized as a directed graph. As one executes a gesture, one's mind transitions between states, generating the appropriate motor commands associated with each state and transition. Since each action is unique, transitions between states are variable. This information is captured within a transition probability matrix $A = \{a_{ij} \mid 1 \leq i, j \leq N\}$ $\left(\sum_{j=1}^{N} a_{ij} = 1\right)$, which represents the probability that an individual will internally transition from state s_i to s_j. Further, depending on the underlying model, one's initial state can be any of the possible model states. This information is captured by the initial probability distribution $\Pi = \{\pi_i \mid 1 \leq i \leq N\}$.

Note, we speculate that the underlying model does not encode motion directly. Imagine one first gesturing a five point star with their hand and then again with their foot. Both arise from the same

internal five point star model, but necessarily each manifests through unique motor actions. So for now, we only claim that the internal model is an abstraction whose exact nature is unknown. Nevertheless, we only need know that the model exists.

We observe the process's output through an input device sampling one's movement. All possible responses are codified into a set of M observation symbols $V = \{v_i \mid 1 \leq i \leq M\}$. Set V, for example, may represent poses or directions of motion. The probability of observing a particular symbol depends on one's state; *e.g.*, we only expect to see significant upward motion near the start of a jump. We capture this information in an emission probability matrix $B = \{b_{im} \mid 1 \leq i \leq N, 1 \leq m \leq M\}$, which is the probability of observing symbol $v_m \in V$ while the system is in state $s_i \in S$.

Putting these components together, one must decide S and V, and then learn Π, A, B from training data to define an HMM $\lambda = (S, V, \Pi, A, B)$. Inferred from our description above, motion comprises two components, which are a series of T observations O and their associated states:

$$O = o_1 o_2 o_3 \ldots o_T \tag{3.15}$$

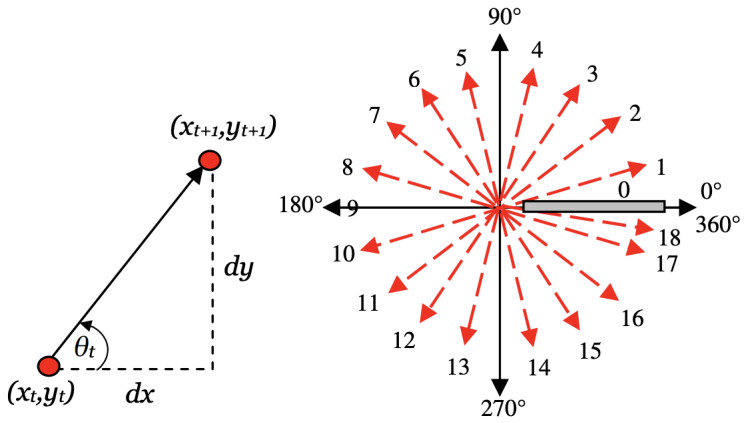
$$Q = q_1 q_2 q_3 \ldots q_T, \tag{3.16}$$

where each observation symbol $o_i \in V$ and state $q_i \in S$. With respect to gesture recognition, our objective is to determine the likelihood $P(O \mid \lambda)$. One simplifying assumption given by an HMM is the Markov assumption—the probability of a state transition depends only on the current state:

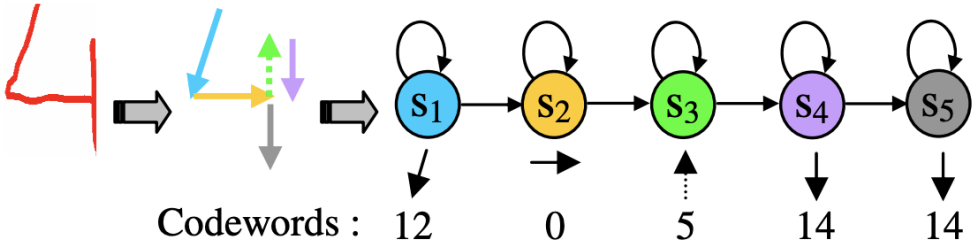
$$P(q_i \mid q_1 \ldots q_{i-1}) = P(q_i \mid q_{i-1}) = A_{i,i-1}. \tag{3.17}$$

And an emission probability similarly only depends on the current state:

$$P(o_i \mid q_1 \ldots q_{i-1}, o_1 \ldots o_{i-1}) = P(o_i \mid q_i). \tag{3.18}$$



Observation Symbol Encoding ©2008 IEEE. Image included with permission.



State Transition Diagram ©2008 IEEE. Image included with permission.

Figure 3.4: Hand gesture HMM approach proposed by Elmezain *et al.* [64]. Top: tracked hand trajectories are converted into direction vectors (left), which is codified by orientation into symbols (right). Bottom: the state transition graph used to recognize the 4 gestures.

These assumptions enable us to use the forward algorithm, a dynamic programming approach that yields the maximum $P(O \mid \lambda)$. Unfortunately, its complexity is high, $\mathcal{O}(TN^2)$. For learning HMM λ, one can use the forward-backward algorithm [16], which is based on the Expectation-Maximization algorithm.

To illustrate some of these concepts, Elmezain *et al.* [64] proposed the HMM hand gesture recognition system illustrated in Figure 3.4. They specifically codify trajectories into symbols that indicate direction of hand movement projected onto the 2D image plane. With a left-right banded (LRB) design, states are topologically arranged left to right, and transitions allow the system to stay in

place or move right one state. On the other hand, Schlömer *et al.* [220] used a left-right HMM, which, being non-banded, allows the systems to skip states. Working with 3D Wiimote accelerometer data, they used k-means clusters on a sphere surface to codify input. These examples illustrate how state topologies and input codification may be somewhat domain specific.

3.3.5 Artificial Neural Networks and Deep Learning

Artificial neural networks take their inspiration from biology, in particular, from the observation that complex neuronal networks learn by correlating data through repeated electrochemical stimulation. In short, "neurons that fire together wire together[2]." In detail, multipolar neurons are composed of a branching dendrite structure that receives data from upstream neurons; a cell body that attends to basic metabolic and homeostasis needs; and a protruding axon whose tail is also a branching terminal structure that transmits information to downstream neurons. These branch-like structures enable individual neurons to communicate with numerous neurons simultaneously though a process called neurotransmisison. When a neuron fires, its axon terminal releases molecular neurotransmitters into the synaptic gaps between itself and other neurons. These neurotransmitter molecules bind to nearby dendrite receptors so as to either inhibit or excite the electric potential across its membrane. When this differential reaches a certain threshold, the neuron fires, releasing its own neurotransmitters onto other neurons.

With respect to learning, synaptic strength increases through long-term potentiation. That is, repeated firing causes biochemical changes including receptor and synapses synthesis, as well as neurotransmitter quantity boosts, making future action potentials more likely to trigger (or inhibit) downstream neurons. In other words, the strength between coupled neurons grows over time with

[2]The exact Donald Hebb quote is "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased [93, p. 64]."

constant excitation. Artificial intelligence researchers have mathematically modeled these neuronal structures and behaviors with varied complexity over the years. An early attempt led to the fairly simple perceptron that one can use for binary classification:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{3.19}$$

where weights $\boldsymbol{w}$ represent synaptic strength, input vector $\mathbf{x}$ represents stimuli or upstream neuron outputs, and bias b modifies the action potential threshold. That is, the perceptron "fires" when the biased inner product surpasses zero, and the input is classified as belonging to the associated class. As implied, however, one can chain together multiple perceptrons or variants of perceptrons into complex hierarchical networks that contain hidden layers. To use these artificial neural networks, one will optimize (learn) network weights, for example, through gradient descent [163], which requires significant quantities of training data.

Note that Equation 3.19 is not differentiable, which is necessary for certain types of learning algorithms that determine weights from training data. A variant of the perceptron called the sigmoid threshold is however differentiable:

$$\sigma(\boldsymbol{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w} \cdot \mathbf{x} + b)}}. \tag{3.20}$$

Fast forward to the present day and deep learning, and we find researchers have made several advances. One such advancement is the introduction of recurrent neural networks (RNN). Unlike traditional neural networks, RNN systems are able to remember previous state information that they combine with new input to make predictions. This formulation allows them to make informed decisions about current events based on prior events, and these systems are therefore amenable to

data with temporal correlations, such as gestures.

Recurrent units are the building blocks of RNNs, and there are many types. In mathematical terms, we write that the hidden state of a recurrent unit at time step t is $\boldsymbol{h}_t = R(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}; \boldsymbol{\theta})$, where $\boldsymbol{x}_t$ is the current input and $\boldsymbol{\theta}$ are trained weight parameters. A popular recurrent unit variant is called Long Short-Term Memory (LSTM) [98]. An LSTM embeds gating logic via neural network layers that decides when and how to update the internal state. Part of this work is also controlled by cell state information that persists over time steps. At each step, an LSTM decides what to forget from the previous step, what to add to the current state, and finally, what to output in the next step. Note that the cell and hidden states are different variables. See [180] for an accessible explanation of LSTM.

A simpler variant known as the gated recurrent unit (GRU) [44] tends to perform as well as LSTM, yet is more efficient. Unlike LSTM, GRU units only propagates hidden state information, not cell state. One such GRU is defined as follows:

$$
\begin{aligned}
r_t &= \sigma\left(\left(W_x^r\, x_t + b_x^r\right) + \left(W_h^r\, h_{(t-1)} + b_h^r\right)\right) \\
u_t &= \sigma\left(\left(W_x^u\, x_t + b_x^u\right) + \left(W_h^u\, h_{(t-1)} + b_h^u\right)\right) \\
c_t &= \tanh\left(\left(W_x^c\, x_t + b_x^c\right) + r_t\left(W_h^c\, h_{(t-1)} + b_h^c\right)\right) \\
h_t &= u_t \circ h_{(t-1)} + \left(1 - u_t\right) \circ c_t
\end{aligned}
\tag{3.21}
$$

A recent example of a gesture recognizer utilizing a GRU based architecture is DeepGRU [155], shown in Figure 3.5. Their system achieves top tier classification performance across a variety of input device types and datasets, including Kinect, pen and touch, and sound wave input. Compared to other DNN architectures, their system has fewer modules and parameters, and is therefore easy

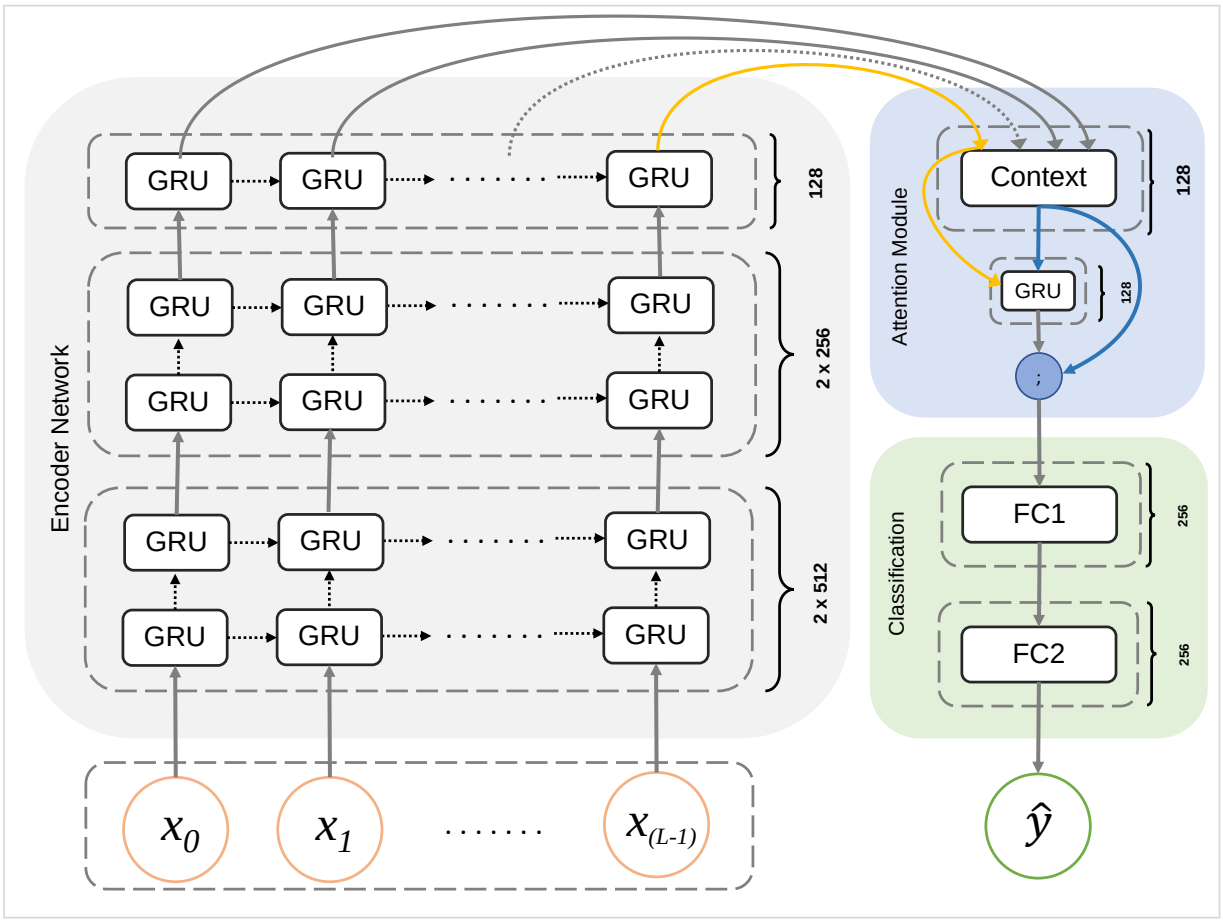


Figure 3.5: DeepGRU architecture. Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer Nature Springer eBook "DeepGRU: Deep Gesture Recognition Utility"by Mehran Maghoumi, and Joseph J. LaViola, Copyright ©2019.

to implement and train.

Downsides to neural networks in general and deep unreal networks (DNN) in particular are obvious. First, they require significant amounts of training data to learn their numerous weight parameters. This can be overcome with synthesis, but customization is still difficult. Training times are also long, which makes online training, necessary for customization, problematic. Last, the barrier to understanding how DNN systems work is high, requiring knowledge of architectures,

node types, optimization, and the loss functions used to optimize DNN networks.

3.3.6 Principle Components Analysis

Principle Component Analysis (PCA) is a statistical technique that transforms data into a new coordinate system in which the axes are ordered by variance. That is, the greatest variance falls onto the first principle component axis, the second greatest variance onto the second axis, and so forth. One approach to gesture recognition with PCA is to learn the transformation for each gesture class, project query samples into the new space, and measure their Euclidean distance from other known samples in the space [126]. Although PCA typically requires enough training data to accurately estimate variance, similiar techniques have been applied to customization.

Escalante *et al.* [65] introduced principle motion components (PMC) for one-shot gesture recognition. In their approach, a video is first converted to a set of energy differences between consecutive frames, and each difference frame is reduced to a grid by averaging all pixels that map to the same cell. The rows of each grid are then concatenated to form a 1D representation of the frame, and all 1D frames are stacked to form a matrix representation of the gesture. Thereafter, PMC learns a model of the gesture by performing principal components analysis using singular value decomposition. They represent each template gesture as a predetermined number of components and perform classification as follows. First, a query video is converted into matrix form as described. The matrix is then transformed using the template principle components, reconstructed by inverting the transformation, and finally the reconstructed matrix is compared to the original matrix. The template with the least reconstruction error is assumed to belong to the same gesture class as the template.

Their work illustrates an example complex customization approach suitable for experienced individuals, but one that is not accessible to all individuals. PCA, for example, is a nontrivial calcula-

tion that may be difficult for some to debug when written from scratch, and data in the transformed space may have a non-intuitive interpretation that is also difficult to visualize. With this example, we illustrate that although an approach supports customization, it does not necessarily align with our goals.

3.3.7 Summary of Common Techniques

We did not cover every gesture recognition approach and certainly not the most complex. Instead, we provided an overview of some commonly employed techniques so that one may compare and contrast with those methods discussed in the next section. It should be understood, we do not believe all machine learning approaches discussed thus far are overly difficult to implement or understand provided one has an appropriate background. We only argue that many who wish to engage in gesture recognition research, user interface prototyping, or indie game development, among other application domains, may have to acquire a significant background in order to understand, implement, and troubleshoot their technique of choice. One may also have to become quite innovative in engineering features appropriate to their circumstance. Further, most methods discussed require a significant amount of training data so that their system can estimate probability distributions. An additional issue not yet discussed is that data collection itself is a tricky proposition (see Chapter 12), and data acquired incorrectly will not yield accurate models. Let us now discuss some gentler, yet still highly effective approaches to custom gesture recognition.

3.4 Keeping It Stupid Simple

In contrast to most methods discussed in the prior section, all present $-family recognizers use straightforward nearest neighbor pattern matching, which is said to be the simplest instance-based

learning method [164]. They transform each training sample into a *template* that serves as a prototypical example of a particular class pattern. Then given a query Q, set of templates $\mathbb{T}$, and dissimilarity measure $f : \mathscr{R}^n \times \mathscr{R}^n \rightarrow \mathscr{R}$, a recognizer will evaluate the likeness of query Q against each template $T_t \in \mathbb{T}$ and infer that query Q belongs to the same class as that of the best matching template:

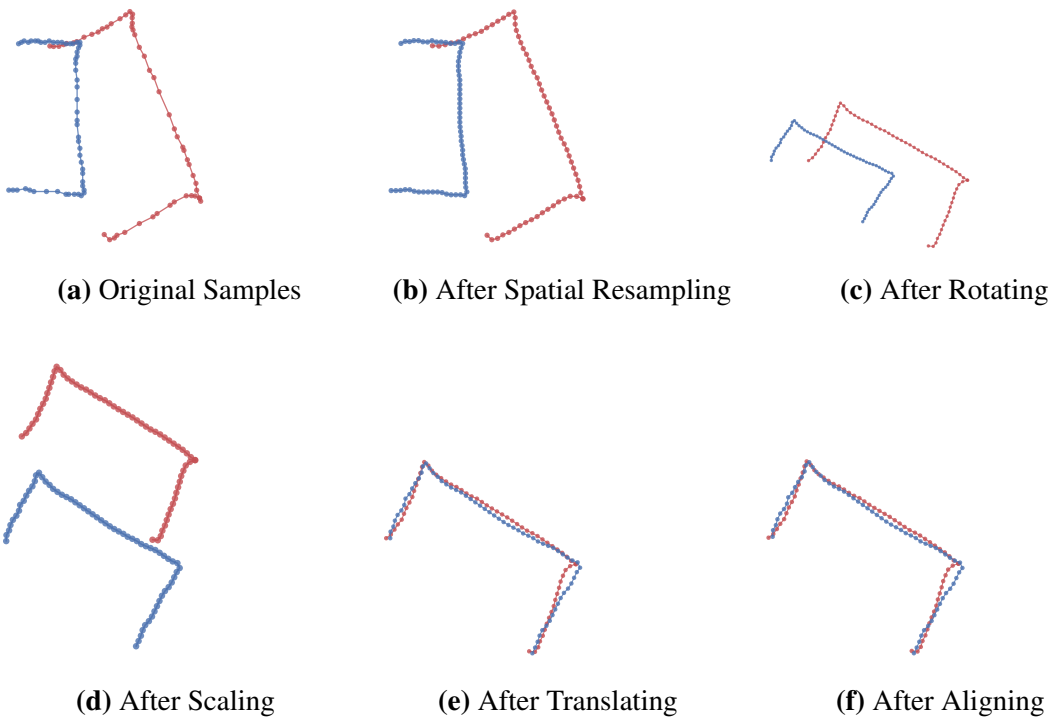
$$\mathbb{T} = \{T_t \mid t = 1 \ldots T_N\} \tag{3.22}$$

$$g_i = G\left(\underset{T_j \in \mathbb{T}}{\operatorname{argmin}} f\left(Q, T_j\right)\right), \tag{3.23}$$

where $g_i = G(T_t)$ specifies template T_t's gesture class ($g_i \in \mathbb{G}$), and $|\mathbb{T}| = T_N$ is the number of templates. How a recognizer transforms, represents, and measures trajectory data defines the difference between each $-family recognizer. Some example common transformation types used in the literature are shown in Figure 3.6.

3.4.1 On Time

In order to address temporal and time series length variability, most approaches use *spatial resampling* to transform a given trajectory into a fixed length series with equidistance spacing between points along the trajectory's arc [8, 9, 96, 143, 235, 238, 248, 250, 253, 267]. Spatial resampling also serves as a low-pass filter that removes some high frequency noise. Vatavu *et al.* [256] explored alternative data preparation techniques, including use of Catmull-Rom splines, which ultimately serves the same purpose. Caputo *et al.* [31] similarly use cubic spine interpolation with resampling as part of their data preparation process. Fuccella and Costagliola [78], on the other hand, use a Douglas-Peucker (DP) line simplification algorithm [55] to select characteristic dominant points that best describe a trajectory.



(a) Original Samples **(b)** After Spatial Resampling **(c)** After Rotating

(d) After Scaling **(e)** After Translating **(f)** After Aligning

Figure 3.6: Handling variations in gesture production.

3.4.2 On Scale and Position

There are a variety of methods for dealing with scale and position variance, yet $-family recognizers most often employ min-max normalization variants. One uses standard min-max normalization to fit a given trajectory into a predetermined bounding volume, *e.g.*, a unit square centered at zero; however, since $1, the usual approach has been to scale trajectories to fit a predetermined volume, and then translate its centroid (not center) to zero. Note that one must take care not to amplify noise, which can happen when motion occurs primarily in one dimension as with a minus sign. In this case, one can instead preserve a trajectory's aspect ratio via uniform scaling [8]. Herold and Stahovich's 1¢ recognizer [96] converts a trajectory into a one-dimensional time series of z-score normalized centroid-to-point measures, thereby addressing position and scale variance simulta-

neously. Another option employed by 3 cent [31] is to scale the trajectory to unit length. To altogether avoid issues with scale and position variability, one can use an alternative data representation. Penny Pincher [238] and Jackknife [235] both work on normalized direction vectors.

3.4.3 On Orientation

Early $-family recognizers such as $1 and $N [8] dealt with orientation variability by rotating the indicative angle to zero. These recognizers also employed a golden-section search (GSS) to iteratively find the optimal query-template alignment. Protractor [143] replaced GSS with a fast closed form solution, and Protractor3D [129] extended this idea to 3D. However, the relatability of Protractor3D's approach has been brought into question since *"[it relies] on quite strange normalization options for gestures"* [31]. Later recognizers including $P [250], Penny Pincher [238], $Q [253], and others did not specifically address orientation variance and yet were able to maintain high accuracy.

3.4.4 On Input Device Type

Most $-family recognizers were designed for 2D input and only a few attempts have crossed over into other domains. Two early examples proposed by Kratz and Rohs for accelerometer data were $3 [128] and Protractor3D [129]. Liu *et al.* brought us uWave [146], a dynamic time warping (DTW) based recognizer also designed for accelerometer input. Although uWave's authors were not motivated by $-family principles, in our opinion, their work conforms to the standard. Caputo *et al.* designed 3 cent [31] for midair hand gestures, and although unspecified, their recognizer may also work with other types of 3D data. Purposefully breaking the input-device-specific design trend, we introduced Jackknife [235], the first device-agnostic custom gesture recognizer that maintains high recognition accuracy across varying input device types—we evaluated Jackknife

over accelerometer [38], full body [63], acoustic [190], and pen [267] data.

3.4.5 On Measuring Dissimilarities

So far we have discussed variabilities in scale, position, orientation, and input, which now brings us at last to dissimilarity measures. $1 uses Euclidean distance to measure the separation between corresponding template-query points, whereas Protractor [143] treats each trajectory as a single point in space and measures the angular inverse cosine distance between aligned trajectories, a technique later employed by $N-protractor [9]. $P ignored time order information inherent in time series data by treating trajectories as point clouds, instead matching points using a greedy weighted Euclidean distance scheme. $P+ [248] improved on $P by incorporating curvature information along with a better matching process. Hausdorff measures [57] have also been considered [247, 250]. Unlike other recognizers that work on points, Penny Pincher measures the inner product between corresponding template-query direction vectors, where each trajectory is defined as a set of directions through space. To better align sequences, Jackknife builds on Penny Pincher by leveraging DTW with an inner product local cost measure.

3.4.6 On the Need for Speed

Researchers have so far improved performance by using pruning techniques, data reduction, and computationally efficient measures. Under the first category, templates are processed in ways that allow recognizers to avoid full computation. Quick$ [208], for instance, uses agglomerative hierarchical clustering to arrange templates into a tree structure, where each parent node contains a reference template and boundary measure—a dissimilarity score encapsulating all children. During a query search, Quick$ prunes any branch whose query-reference dissimilarity score is greater than both the boundary and best found solution, since it is guaranteed by the bound that no better

solution exists in the subtree[3]. Jackknife and $Q take a different approach. Both use lower bounding techniques to avoid a full evaluation of those templates that cannot improve on the best found match by first employing a cheap lower bound calculation. Similar in spirit, Zhang *et al.* [278] use nonlinear embedding [105, 106]. Specifically, they map templates into a 2D space based on the mean and variance of all trajectory points and components combined. Due to the Cauchy-Schwarz inequality, the Euclidean distance between a query and template in its original space is bounded by a factor of their distance in the 2D embedded space, which one can again use to prune inferior matches[4]. Vatavu introduced the 1F accessory feature [246] that maps templates and queries down to a one dimensional scalar space (example statistical features include duration, trajectory length, and total curvature). Templates are sorted by their 1F feature, and a recognizer fully evaluates only those templates that fall within the local neighborhood of the query. These techniques are successful in accelerating recognizer performance, but also add complexity that we hope to avoid by using data reduction and fast measures.

Data reduction techniques transform data into small meaningful forms, thereby lowering the computational work involved at query time. Nonlinear embedding and 1F are indeed example reduction techniques, but measures over the transformed data cannot sufficiently discriminate gesture classes, which is why they are used for pruning rather than classification. Spatial resampling, however, is quite effective when the resampling rate N is sufficiently low, *i.e.*, $N \ll |T_t|, \forall T_t \in \mathbb{T}$. It has been shown that one can achieve high accuracy with as few as 6 samples per trajectory when using the Euclidean distance or angular inverse cosine measures for 2D data [245], and similar results over varying measures were also reported for 3D data [247]. The Douglas-Peucker (DP) line simplification techniques (used by Fuccella and Costagliola [78]) is an alternative option that, as previously discussed, finds those points that best describe the shape of a line. Given the variability

[3]Quick$ assumes that the $1 measure obeys the triangle inequality, which is not proved.

[4]This is one example of an advanced technique that reduces to an easy to implement and understand approach. So although $-family principles are straightforward, they can originate from expert knowledge.

of gesticulation, DP selected points are unlikely to correspond well in Euclidean space and so one must use alternative measures. Other work has looked at how to reduce the training set size, which may be necessary when over time an application logs a large number of training samples. We, for example, used random mutation hill climb (RMHC [224]) to select those templates that best separate the custom gesture classes [191]. They show high accuracy is possible when only a small, intelligently selected subset of data is used to train the recognizer.

Complementary to pruning and data reduction, computationally efficient measures also impact performance. Li [143] addressed $1's bounded rotation alignment strategy by (among other things) introducing a closed-form solution that $1 had solved via an iterative optimization process. Penny Pincher used an inner product calculation that avoids system calls to inefficient math functions like square root and arccosine, to achieve nanosecond performance. Jackknife uses a Sakoe-Chiba band [218] to not only prevent pathological warping, but significantly speed up its DTW-based calculation.

3.5 Finding Gestures in Continuous Input

Gesture spotting and segmentation is said to be as difficult as gesture recognition itself [261], and as with recognition, there are countless approaches practitioners use to spot and segment continuous data. Differences, in part, stem from which question one sets out to answer. Let $X_{a,b}$ be a subsequence of X:

$$X_{a,b} = (\boldsymbol{x}_i \mid 1 \leq a \leq i \leq b \leq |X| \ \wedge \ \boldsymbol{x}_i \in X)\,. \tag{3.24}$$

Example questions we can ask follow: At time i, what is the probability that $\boldsymbol{x}_i \in X$ is a gesture

boundary, regardless of class? Given history $X_{1,i}$, to what gesture class does sample $\boldsymbol{x}_i \in X$ most likely belong? At time j, what $i \leq j$ most likely maximizes $X_{i,j}$'s similarity to class g?

Escalera *et al.*'s survey paper [66] on multi-modal gesture recognition discusses segmentation and identifies two main approaches: direct and indirect. A second survey by Weinland *et al.* [261] uses a slightly different organization: boundary detection, sliding windows, and grammar concatenation. Between these surveys, direct methods entail boundary detection and windowing, whereas indirect methods correspond to higher-level grammars. Therefore, we report on direct and indirect segmentation strategies, though our goal here is to inform the reader about commonly employed techniques and discuss their relation to our design goals. For more information, please refer to the surveys.

3.5.1 Direct Methods

Direct methods use heuristics or examine physical characteristics of motion to localize endpoints, such as velocity, acceleration, and poses. However, these methods tend to be unreliable in the presence of complex interactions [261]. For example, to detect gesture boundaries in dance sequences, Kahol *et al.* [116] hierarchically track velocity, acceleration, and mass of various segments that they coalesced into a single force, kinetic energy, and momentum result. They then look for minima in the total body force signal to guide subsequent analysis in deciding boundaries. Kang *et al.* [118] similarly look for abnormal velocities, static poses, and severe curvatures to help detect candidate gesture boundaries in video games. Ye *et al.* [272] use curvature to segment a single stroke gesture into constituent strokes. Chen *et al.* [43] use energy and root mean squared (RMS) to segment electromyography (EMG) signals in real time, which is a common measure practitioners employ when working with EMG data. Arn *et al.* [11] used generalized curvature analysis to segment user gestures from continuous Kinect skeletal data. Their system is able to

segment main motions from transitions. Specifically, they infer regions of high curvature map to transition phases, whereas main motions have lower curvature. This can be seen as the inverse of energy based segmentation, where low energy moments are treated as candidate segmentation points. Other examples include [222, 259].

Perhaps the most prevalent technique used for segmentation is the sliding window, which continuously evaluates a fixed subset of the most recent input. Windowing is attractive because of its simplicity, as one only needs to keep a small history buffer that the system continuously feeds to an underlying recognizer. Sliding windows have been used in data glove [152], body-worn inertial sensors [115], and vision based body and tracking [228] gesture recognition systems. One problem with windowing, however, is that one must decide on an appropriate window size. This issue can be resolved by learning appropriate window sizes with training data [152] or by using multiple windows at the expense of greater computation.

Keogh *et al.* [121] presents an overview of several techniques for time series segmentation, including windowing, top-down, and bottom-up. The former processes an entire time series at once and splits it repeatedly based on a given measure (such as energy) until either it reaches a set number of segments or no further splits are possible. The latter technique conversely groups sequences together that have similar properties until a certain criteria is met. Both top-down and bottom-up are traditionally offline segmentation approaches, which limits their utility compared to sliding windows, despite being generally more accurate. Keogh *et al.*, however, propose the sliding window after bottom-up (SWAB) technique, which builds small segments that can be merged in the future.

Different from most other direct approaches, Vatavu *et al.* [254] proposed a hierarchical recognition approach based on the integral absolute curvature of 2D gestures, which they proved to be scale invariant. With appropriate smoothing, a gesture's curvature is sufficiently consistent as shown in Figure 3.7, and can be used to segment gesture candidates for further analysis. One can

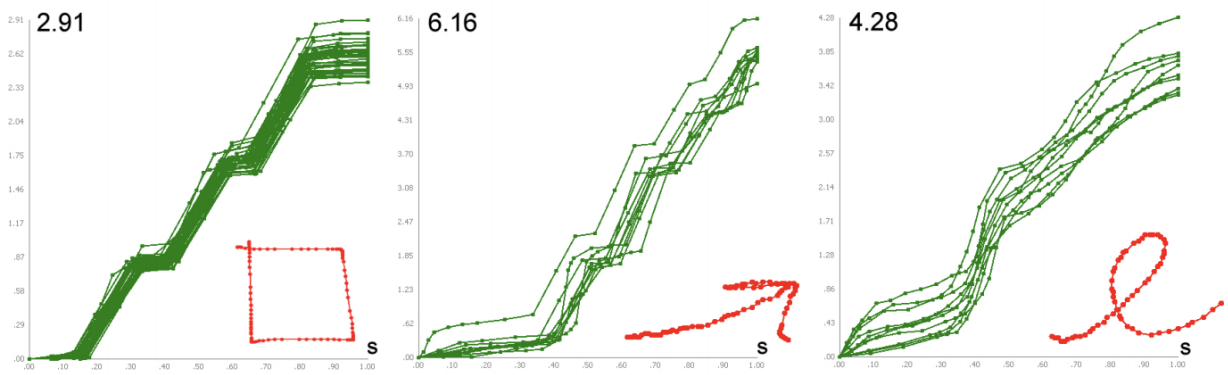


Figure 3.7: Integral of absolute curvature for three unique gestures, from [254]. Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer Nature Springer eBook "Multiscale Detection of Gesture Patterns in Continuous Motion Trajectories" by Radu-Daniel Vatavu, Laurent Grisoni, and Stefan-Gheorghe Pentiuc, Copyright ©2010.

learn the range of the integral of absolute curvature for a given gesture from a modest amount of training data.

Simple, heuristic (rule-based) segmentation is also common, whereby a system uses easy to detect criteria in order to delineate gestures from other user activities. Using a Kinect, Kristensson *et al.* [130] defined an "input zone" for hand gestures where users could interact with their application. Movement into and out of the zone therefore segmented the continuous input stream. Gesture Watch [123] instead uses a hardware sensor to detect when one raises their wrist, at which time the system begins to record input from other sensors until one again lowers their wrist, thereby triggering recognition.

3.5.2 Indirect Methods

Indirect methods are like mini-recognizers that score continuous data until a possible gesture boundary is detected. One example is Alon *et al.*'s spatiotemporal HMM recognizer [5] as well as

Yin and Davis's three part HMM [273]. The latter approach divides a gesture into three parts—the pre-stroke (setup), nucleus (main activity), and post-stroke—and learns separate models for each. Other examples include artificial neural networks [172] for hand gestures and conditional random fields for sign language [271]. Yin and Davis [273] trained an HMM to detect hand gestures from salience maps extracted from RGB-D images. Wang *et al.* [260] made use of an HMM for segmenting atomic gestures from streams of human actions in recorded videos.

For a more recent example, consider uDeepGRU, which took first place in the 2019 3D shape retrieval contest (SHREC'19) for continuous hand gesture recognition. uDeepGRU is an end-to-end deep neural network designed for recognizing gestures in an unsegmented stream of data. With gated recurrent units (GRU) [45] as the main building block, uDeepGRU takes as input per-frame raw direction vector features and outputs class conditional probabilities. Feature extraction and implicit segmentation are performed jointly without relying on the data of future time steps. These properties make uDeepGRU suitable for online recognition of gestures, but nontrivial to understand and implement.

Conversely, one approach suitable of custom gesture recognition is continuous dynamic programming (CDP) [179], which relaxes a DTW constraint so that continuous recognition is possible. Similarly, Sakurai *et al.* [219] use a modified DTW algorithm, called SPRING, to continuously segment single or multidimensional data streams. The two modifications they proposed were star-padding, which reduces the number of matrices one needs to calculate per candidate template, and sub-sequence time warping, which adds additional information to each individual cell of the DTW matrix. These two modifications dramatically reduce spatial and temporal complexity in time series matching. In our view, SPRING and CDP are very similar.

3.5.3 Accept-Reject Criteria

A continuous gesture recognizer evaluates input as it arrives, and for each new sample, the recognizer must decide whether or not to inform the system of a possible gesticulation. This decision comprises the *accept-rejection criteria*—the set of conditions one must satisfy in order to send notification. Typically, scores that fall below a per class *rejection threshold* are discarded, whereas sufficiently high scores result in further analysis. What happens thereafter is nuanced, application specific, and rarely discussed in the literature, despite having a significant impact on recognition accuracy. When descriptions are provided, they often lack detail, appear ad-hoc, or seem to fit the data at hand [118, 268, 273]. To illustrate with a few examples, some approaches require that a duration of time passes between subsequent recognition events as a way to remove unintended non-gesture movements and potential noise [28, 190]. Similarly, a recognizer may require that an evolving pattern scores well over multiple consecutive frames prior to acceptance, so as to circumvent false detection from arbitrary spikes in noisy data [116]. Recognizers may further purge [5, 24] or keep [23] prior data and internal state information upon detection, which impacts subsequent signal processing. One may use a low sampling rate to stabilize results or interpolate across multiple frames [24].

3.6 Synthetic Data Generation

The paucity of correctly labeled training data is a common problem in the field of pattern recognition [171]. Crowdsourcing can help alleviate this issue, although with potentially high cost. Alternatively, one may synthesize new data from that which is already available. Synthetic data generation (SDG) has been used successfully in many fields, including human pose recognition [223], digital forensics [80, 149], information retrieval [82, 211] and handwriting recognition of ancient text [72], as well as in generating [12, 68] and collecting [153] large data sets.

Early examples of SDG in gesture and handwriting recognition include works by Ha and Bunke [89] and thereafter, a number of researchers have also attacked this problem as reported in Elanwar's survey [62]. One key difference in their approaches relates to whether the data is digital ink or images of symbols. Methods that work on ink can broadly be divided into two categories: those that replicate feature distributions of the population (such as pen-lifts and velocity) and those that apply perturbations to the given data. A third option involves the interactive design of procedurally generated gestures, such as that provided by Gesture Script [148].

Concerning image based techniques, Helmers *et al.* [94] proposed an approach to produce synthetic text in which isolated samples of an individual's handwriting are concatenated together to form synthetic handwritten text. Ha and Bunke [89] along with Cano *et al.* [29] leveraged various image transformation operations (*e.g.*, erode, dilate, *etc.*) to produce variations of image samples of handwritten characters. Varga *et al.* [242, 243] used randomly generated geometrical transformations such as scaling and shearing lines of handwritten text to produce new synthetic lines. Such transformations have also been applied toward the creation of synthetic CAPTCHAs [240]. Lee *et al.* [139] generated synthetic Korean characters using Beta distribution curves while Varga *et al.* [244] used splines to generate handwritten English text. Caramiaux *et al.* [33] generated synthetic samples drawn from Viviani's curve to produce controlled data to evaluate their proposed adaptive gesture recognizer. Dinges *et al.* [54] used active shape models which rely on the linear combination of covariance matrix eigenvectors built for each class of shapes to create synthetic handwritten text. In the remainder of this subsection, we take a closer look at some common approaches.

3.6.1 Perlin Noise

Davila *et al.* [49] used Perlin noise maps [188], a well-known technique in computer graphics for producing natural looking synthetic textures, to generate synthetic math symbols. Each Perlin map

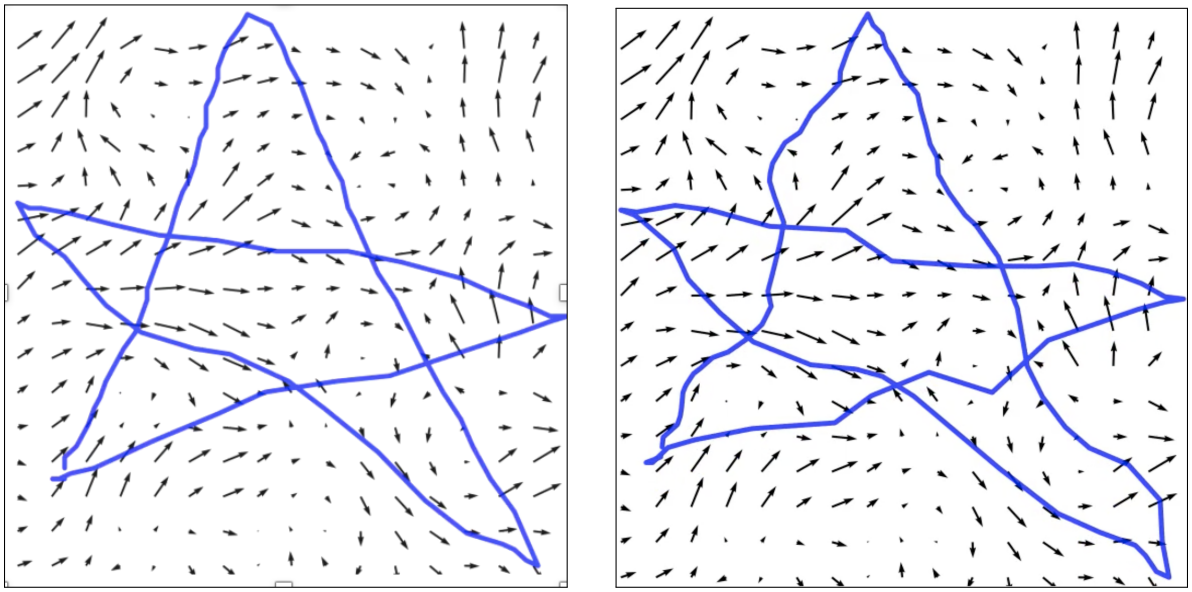


Figure 3.8: Perline noise synthetic data generation example. A gesture is placed into a vector field (left) and warped into a variant of itself (right).

consists of a grid of points. The number of points defines the resolution of the map. A gradient direction is assigned to each point and random noise is generated based on the direction of the gradient. Synthetic samples are created by coinciding the gesture's stroke points on the noise map and moving each stroke's points along the grid's gradient direction. We illustrate this process in Figure 3.8.

3.6.2 Sigma-Lognormal Model

Although numerous models have been proposed to describe human movement, the Kinematic Theory of rapid human movement [193, 194] and its associated Sigma-Lognormal ($\Sigma\Lambda$) model [195] has been shown to have superior performance in modeling human movement, and has been successfully applied to a large range of applications. The $\Sigma\Lambda$ equations (including Equations 3.25 and 3.26) attempt to model the complex interactions of a neuromuscular network executing an

action plan. That is, a stroke is described by a set of overlapping primitives connecting a series of virtual targets [141], where each primitive is described by a lognormal equation. Formally, the velocity profile of a trajectory is given by:

$$\vec{v}(t) = \sum_{i=1}^{N} \vec{v}_i(t) = \sum_{i=1}^{N} D_i \begin{bmatrix} \cos\phi_i(t) \\ \sin\phi_i(t) \end{bmatrix} \Lambda(t; t_0, \mu_i, \sigma_i^2), \tag{3.25}$$

which is the vectorial summation of N primitives. Each primitive is a four parameter lognormal function scaled by D_i and time shifted by t_i, where μ_i represents a neuromuscular time delay and σ_i, the response time. The angular position of a primitive is also given by:

$$\phi_i(t) = \theta_{s_i} + \frac{\theta_{e_i} - \theta_{s_i}}{2} \left[1 + \text{erf}\left(\frac{\ln(t_i - t_0) - \mu_i}{\sigma_i \sqrt{2}} \right) \right], \tag{3.26}$$

where θ_{s_i} and θ_{e_i} are starting and ending angles of the ith primitive. A parameter extractor such as that described by Martín-Albo *et al.* [160] is used to find the individual primitives and their associated model parameters for a given stroke. Perturbations to model parameters create realistic variations in the trajectory and can be used to create synthetic gestures, such as for whiteboard note generation [71]. Closer to our interests, Leiva *et al.* [141] recently provided the community with their Gestures à Go Go (G3) web service that generates synthetic samples from real data using the Kinematic Theory. They were also able to show that $-family recognizers trained with only synthetically generated samples could perform as well as recognizers trained with only human samples.

3.7 Parametric Curves

Parametric curves are a common way to model gestures, where functions of independent variables or parameters define the trajectory. Consider, for example, a time-dependent circular gesture with radius r, phase shift θ, and angular frequency ω. We can define this 2D gesture parametrically as:

$$\boldsymbol{p}(t) = \begin{bmatrix} r\cos(\omega t + \theta) \\ r\sin(\omega t + \theta) \end{bmatrix}. \tag{3.27}$$

from which we can generate a trajectory to train a recognizer. We might further vary the parameters r, ω, and θ to generate a distribution of synthetic circles.

Another example involves Bézier curves. A linear Bézier curve is defined by two control points $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ whose linear interpolation results in a straight line:

$$\boldsymbol{b}(t) = \boldsymbol{p}_0 + t(\boldsymbol{p}_1 - \boldsymbol{p}_0) \tag{3.28}$$

$$= (1-t)\boldsymbol{p}_0 + t\boldsymbol{p}_1, \tag{3.29}$$

where $0 \leq t \leq 1$, and $t = 0$ and $t = 1$ correspond to the first and last control points of the curve. Suppose now that we have an additional control point and two linear Bézier curves—b_0 interpolates over p_0 and p_1, and b_1 interpolates over p_1 and p_2. A quadratic Bézier curve is then the interpolation of $\boldsymbol{b}_0$ and $\boldsymbol{b}_1$:

$$\boldsymbol{b}(t) = (1-t)\boldsymbol{b}_0 + t\boldsymbol{b}_1 \tag{3.30}$$

$$= (1-t)^2\boldsymbol{p}_0 + 2(1-t)t\boldsymbol{p}_1 + t^2\boldsymbol{p}_2. \tag{3.31}$$

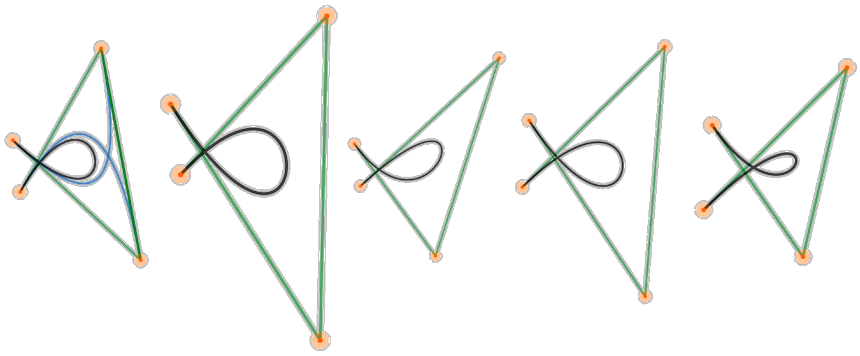


Figure 3.9: Synthetic alpha α gestures generated by perturbing cubic Bézier control points. All instances show the linear Bézier curves in green, but only the first (seed) sample also shows the intermediate quadratic Bézier curves in blue.

Continuing on as such, the general form of an n-th degree Bézier curve constructed from $n+1$ control points is:

$$\boldsymbol{b}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \boldsymbol{p}_i, \tag{3.32}$$

where $\binom{n}{i}$ is the binomial coefficient. In other words, each control point i is weighted by the i-th Bernstein basis polynomial. And in order to generate synthetic variations of a trajectory defined as a Bézier curve, one only needs perturb its control points, as illustrated in Figure 3.9.

3.8 Conclusion

A common theme we saw throughout our gesture recognition and associated techniques review is that many approaches require significant quantities of training data to facilitate learning and parameter tuning. This is true for low-pass filter parameter selection, distribution estimation, and model optimizations. Often, techniques are non trivial, require application specific domain knowledge, use advance mathematical concepts, and benefit from prior practitioner experience. We saw, for

example, that one who is familiar with an application domain might hand craft features amenable to patterns he or she expects to encounter in that domain. Though to give engineered features power, one may have to learn its distribution and employ it within a complex pattern recognition framework such as SVM. Thereafter, through trial and error, or via an application of hard earned knowledge, one may further engage in hyper parameter tuning to squeeze greater performance from their machinery. These issues run contrary to custom gesture recognition. To rapid prototyping. To straightforwardness.

We are interested in customization for those reasons discussed in our foundations chapter. Customization by definition requires that our proposed methods must work with limited training data; namely, we must achieve high performance with only one or two training samples per gesture class. We turn to nearest neighbor template matching for this reason and because template matching techniques offer generally straightforward solutions, satisfying, at least to an extent, our accessibility requirement. In this work, we primarily leverage direction vector sequences rather than features or points because they provide us with an intuitive representation of human motion, and are invariant to scale and position differences, two common sources of variability. Dot product operations on vectors are also serendipitously efficient, involving only multiplication and addition.

We improve on the state-of-the-art in customization by combining elastic matching methods (DTW and CDP) with motion-based direction vector and optional auxiliary measures that improves gesture class separation. The former addresses temporal variability, whereas the later simply improves accuracy. The combination thereof, for the first time that we are aware of, enables fast, efficient, device-agnostic custom gesture segmentation and recognition. Except for Pitch Pipe, each component of our pipeline is, in our view, relatively straightforward, requiring only foundational mathematical knowledge. And although we use dynamic programming, we believe one can understand our approach without prior knowledge of dynamic programming.

In the next chapter, we introduce Pitch Pipe, an automatic technique for selecting low pass filter parameters in a gesture recognition context. Although Pitch Pipe is our most complex pipeline component (because of its reliance on Fourier transforms), it does support customization. Moreover, unlike other filter parameter selection approaches described at the start of this chapter, Pitch Pipe requires zero training data.

CHAPTER 4: PITCH PIPE

(Domo arigato, Mr. Roboto)
Thank you very much Mr. Roboto

(Domo arigato, Mr. Roboto)
For doing the jobs nobody wants to

Styx
Mr. Roboto

Noise from biochemical processes, hardware sensors, digitization, and estimation errors has potential to corrupt motion data embedded in a signal. Perturbations range from small jittery responses in accelerometer data to collapsed skeletal structures from erroneous RGBD analysis. These errors increase variance and can harm recognition accuracy since gestures will measure further apart in their feature space relative to clean signals. It is therefore necessary, in certain contexts, that one filters their input data before engaging in gesture classification work. However, tuning a low-pass filter without prior experience or domain knowledge is a non-trivial task. In this chapter, we discuss a novel, online, low-pass filter wrapper method called Pitch Pipe that automatically calibrates a given filter. Pitch Pipe eliminates the need to collect training data and is device-agnostic.

4.1 Preliminaries

In this section, we discuss digital signal processing (DSP) fundamentals One can describe an arbitrary time series $\mathbf{x} = \{x_0,\ x_1,\ \ldots,\ x_{N-1}\}$ as the sum of N unique sinusoids oscillating at N different frequencies. Fourier coefficients are the complex numbers that encode each sinusoid's amplitude and phase. To recover Fourier coefficients $\mathbf{X} = \{X_0,\ X_1,\ \ldots,\ X_{N-1}\}$ from signal $\mathbf{x}$, one can employ

the discrete Fourier transform (DFT):

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-2\pi i k \frac{n}{N}}, \tag{4.1}$$

where X_k is frequency k's Fourier coefficient, i is the imaginary unit, and n indexes each sample (thus, n/N corresponds to time). Two quantities of interest that Pitch Pipe monitors are amplitude A_k and power spectral density (PSD) ϕ_k. Both of these values can be derived from Fourier coefficients:

$$A_k = \frac{1}{N} |X_k| = \frac{1}{N} \sqrt{\mathrm{Re}(X_k)^2 + \mathrm{Im}(X_k)^2}, \tag{4.2}$$

and:

$$\phi_k = \frac{1}{N} |X_k|^2 . \tag{4.3}$$

Sinusoids in the real domain comprise both positive and negative frequencies (which we exploit in Equation 4.11). Due to Euler's formula, we have:

$$A \cos\left(2\pi k \frac{n}{N}\right) = \frac{A}{2} \left[e^{2\pi k \frac{n}{N}} + e^{-2\pi k \frac{n}{N}} \right], \tag{4.4}$$

where k and $N-k$ are the positive-negative frequency counterparts. Consequently, the Nyquist frequency $f_s/2$ is the maximum frequency we can observe, where f_s is the input device's sampling rate. In this work we set $N = f_s$ or $N = f_s + 1$ when the rate is odd.

4.1.1 Sliding DFT

Continuous input requires ongoing analysis, yet Fourier transforms are computationally taxing, even when using efficient techniques such as the $\mathcal{O}(n \log n)$ faster Fourier transform [225]. Alterna-

tively, one can use a sliding DFT (SDFT) [109, 110] to incrementally calculate relevant coefficients via the following constant time $\mathcal{O}(1)$ update function:

$$S_k(t) = e^{2\pi i k \frac{1}{N}} \left[S_k(t-1) + x_t - x_{t-N} \right]. \tag{4.5}$$

This gives us the Fourier coefficient X_k at time t — signal $\mathbf{x}$'s DFT over the most current N samples. We therefore only require an N-element circular buffer to effectively use this technique. Practitioners should also be aware that other advanced methods exist, such as the modulated SDFT [58] designed to address accumulated rounding errors. In our tests, however, we found the standard SDFT to be sufficient.

4.1.2 Welch's Method

Since motion and noise are random processes, amplitude and power estimates taken from a single DFT are unreliable. A better approach is to partition the input into N-element subsequences whose DFTs are averaged together. We find for Pitch Pipe that plain averaging works well when estimating amplitude, but breaks down for PSD-based noise estimates due to spectral leakage.

Specifically, human motion occurs over a continuum of frequencies, not just on input device harmonics, and so non-integer frequencies discretely sampled "leak" into neighboring bins as illustrated in Figure 4.1. We see in this example that a noisy 1 Hz signal is problem free, but spectral leakage from the ½ Hz sinusoid hides the noise floor we wish to observe. A good solution to this problem is to estimate the power spectrum by averaging together overlapping *windowed* DFTs, i.e., to use Welch's method [114, 263]. The windowed DFT follows:

$$X_k = \sum_{n=0}^{N-1} x_n \, w_n \, e^{-2\pi i k \frac{n}{N}}, \tag{4.6}$$

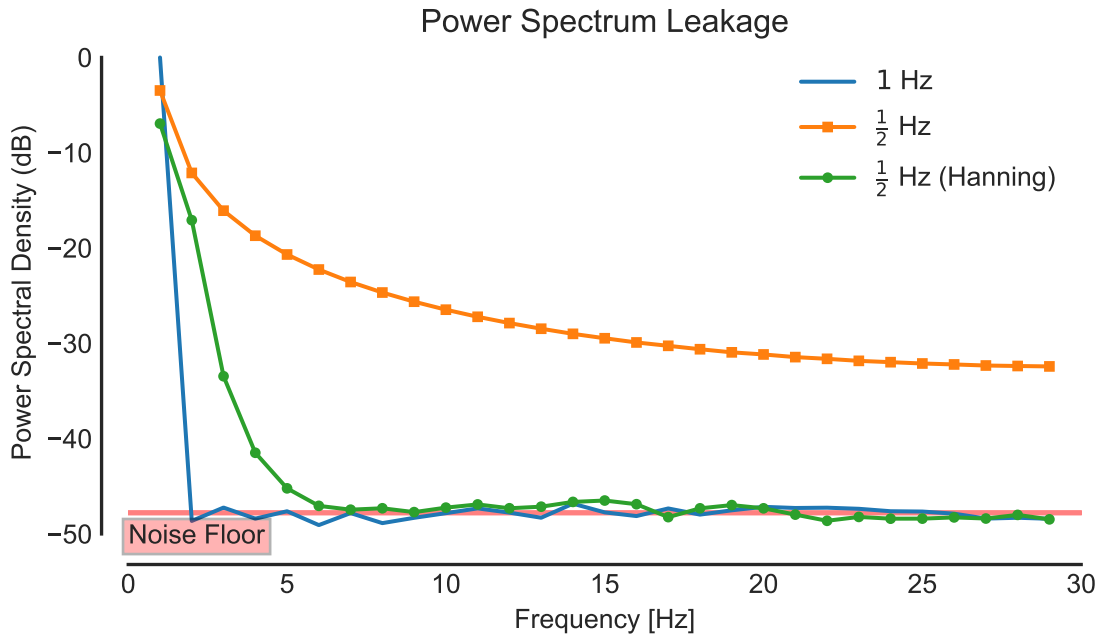


Figure 4.1: Example of a 60 Hz input device sampling a noisy $1/2$ and 1 Hz signal that is analyzed with a 60 point sliding DFT. Power of the 1 Hz frequency is captured entirely in its associated bin, whereas the $1/2$ Hz signal leaks over and pulls the full spectrum high. However, one can use a Hanning window to significantly reduce leakage and drop high frequency power back to the noise floor.

where w_n weights sample x_n, resulting in a modified PSD ϕ_k as follows:

$$\phi_k = \frac{1}{W} |X_k|^2 . \tag{4.7}$$

and:

$$W = \sum_{n=0}^{N-1} (w_n)^2 . \tag{4.8}$$

There are a number of window functions from which one may choose, but we opt for the Hann window because of its popularity and efficient computation (upcoming Equation 4.11):

$$w_n = \frac{1}{2} \left[1 - \cos \left(\frac{2\pi n}{N-1} \right) \right] . \tag{4.9}$$

Referring back to Figure 4.1, one can see that with Hann windowing, spectral leakage is significantly reduced, and we have at least regained access to the noise floor. Because of windowing, however, one can no longer use SDFT (Equation 4.5) directly. Instead, we require three transforms [109]:

$$\mathbb{S}_k(t) = \frac{1}{2}S_k(t) - \frac{1}{4}S_{k-1}(t) - \frac{1}{4}S_{k+1}(t). \tag{4.10}$$

Though observe that when the sampling rate is even and $k = N/2$, frequencies $k \pm 1$ are merely complex conjugates of each other. Thus, Equation 4.10 reduces to:

$$\mathbb{S}_q(t) = \frac{1}{2}S_q(t) - \frac{1}{2}\mathrm{Re}\left[S_{q-1}(t)\right], \tag{4.11}$$

where q refers to the Nyquist frequency $f_s/2$, and $\mathrm{Re}(z)$ denotes the real part of complex number z. We now have enough information to motivate and describe Pitch Pipe.

4.2 Pitch Pipe: The Low-pass Filter Auto-tuner

In this section, we introduce our on-line, automatic low-pass filter parameter selection technique Pitch Pipe.

4.2.1 Motivating Example

Consider time series $\mathbf{x} = \{x_0, x_1, \ldots\}$ defined as the white noise corrupted superposition of two sinusoids:

$$x_n = \sum_{k=1}^{2} A_k \cos\left(2\pi k \frac{n}{f_s}\right) + \varepsilon_n, \tag{4.12}$$

where n is the sample index, A_k is the sinusoid's amplitude at frequency k, f_s is the sampling rate (n/f_s is time), and $\varepsilon_n \sim \mathcal{N}\left(0, \sigma^2\right)$ is the independent and identically distributed error term. Our

objective is to remove noise from signal **x** via a low-pass filter so as to minimize the root-mean-square error. In doing so, we decide to leverage the ubiquitous exponential moving average, which then requires us to select smoothing parameter $\alpha \in [0,1]$. But what is the best setting? We will see that it depends.

Let the input device sample rate be set to $f_s = 60\,\text{Hz}$. Knowing that 2 Hz is the maximum frequency embedded in our signal allows us to naïvely select a 3 Hz cutoff, $\alpha \approx 0.27$ per [225]. As it turns out, however, optimal α actually depends on the strength of information and noise in the signal, not just on the frequency band carrying relevant information. To illustrate, Table 4.1 lists the average optimal EMA smoothing parameter minimizing RMSE over varying conditions. Each condition was replicated ten times over 30 second synthetic signals[1], and we found that each case required a unique solution, all of which are well above our naïve choice.

These results teach us a few important lessons. First, a simple low-pass filter cutoff approach is inadequate given that the same frequencies are present in each condition, yet each case tunes to a different level; that is, simply knowing which frequencies are present in a signal is not enough. Second, increasing a frequency's amplitude affects the optimal parameter, but not consistently across

Table 4.1: Optimal EMA smoothing parameter settings for varying amplitude and noise levels per Equation 4.12. Notice that each condition requires a unique setting.

A_1	A_2	$\sigma^2 = 0.10$		$\sigma^2 = 0.01$	
		μ	(σ)	μ	(σ)
1	1	0.59	(0.0070)	0.84	(0.0040)
1	2	0.73	(0.0066)	0.92	(0.0047)
2	1	0.65	(0.0093)	0.88	(0.0053)
2	2	0.75	(0.0078)	0.93	(0.0050)

[1] A pairwise Holm-Bonferroni [100] corrected independent samples t-test confirmed that all solutions were unique (all $p < .001$).

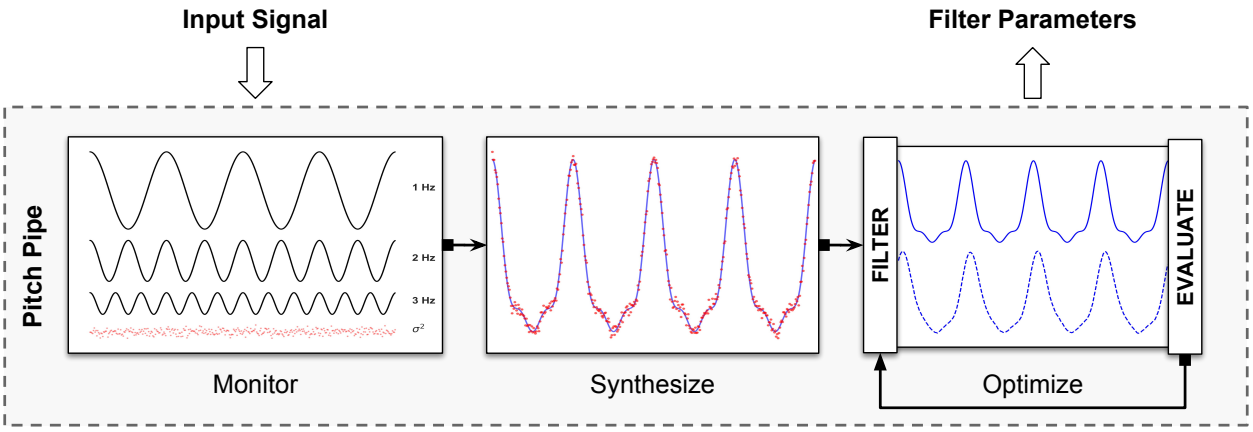


Figure 4.2: Pitch Pipe's three step pipline: First, we monitor the input device signal and estimate low frequency amplitude averages (Equation 4.13) as well as noise variance (Equation 4.14). From these estimates, we generate a noisy synthetic signal (Equation 4.16) and tune the filter until its output matches the synthetic ground truth.

frequencies, *e.g.*, study the effect of doubling A_1 versus that of A_2. Third and perhaps most obvious is that noise strength impacts parameter selection — higher noise variance requires more aggressive filtering. We use these observations to help drive the design of our auto-tuner.

4.2.2 Pitch Pipe

In the analysis of our motivating example, we saw that frequency strength and noise both impact filter parameter selection, which one can easily estimate using DSP techniques. What perhaps is less clear is how one might tune their filter after acquiring such information. If user trajectories were precisely known, then tuning would either be unnecessary or a trivial matter of fitting a filter to the data, but outside of interactive methods, ground truth is unknowable.

Our key insight is that with reliable estimates, one can generate synthetic signals in the likeness of reality — those that possess similar information content and noise. One can then tune their filter using synthetic ground-truth-known sequences and expect similar performance on real data.

This approach is not entirely unlike tuning a guitar by generating sound through a pitch pipe and adjusting string tension to match its resonance. Motivated by our above observations, we designed the three step auto-tuner depicted in Figure 4.2:

1. Estimate low frequency amplitudes and noise;
2. Generate a synthetic signal based on said estimates; and
3. Optimize one's filter using the synthetic signal.

In the remainder of this section, we provide additional information on each step.

Step 1a (Estimate Amplitude): To estimate frequency k's amplitude $\hat{A}_k$, we average together all SDFT results from the start of time (Equation 4.5), excluding only the first N samples used to prime the buffer:

$$\hat{A}_k = \frac{2}{N(T-N)} \sum_{t=N}^{T} |S_k(t)| . \tag{4.13}$$

Note that the magnitude is doubled to account for the positive and negative frequency contribution, which saves us from having to monitor two frequencies (k and $-k$) [225].

Step 1b (Estimate Noise): Frequencies void of information that would otherwise carry zero power are nevertheless corrupted by noise. In human computer interactions, these are frequencies typically above 10 Hz [276] and are the frequencies one should monitor to estimate the noise floor. Since we assume zero-mean Gaussian noise, we need only to approximate variance σ^2. And because PSD $\phi_k = \sigma^2$ [229], we can use Welch's method with a Hann window (Equation 4.11) to

estimate variance:

$$\hat{\phi}_q = \hat{\sigma}^2 = \frac{1}{W(T-N)} \sum_{t=N}^{T} \left| \mathbb{S}_q(t) \right|^2. \tag{4.14}$$

We specifically choose to monitor only the Nyquist frequency for two reasons. First, per Equation 4.11, we are able to leverage a sliding DFT over two frequencies rather than three, which reduces the computational burden. Second, when the sampling rate is low, spectral leakage impacts the Nyquist frequency least. However, if one is compelled, and computational power is plenty, and the sampling rate is high, he or she may choose to monitor additional frequencies and average the individual PSD results into a single $\hat{\sigma}^2$ estimate.

Step 2 (Synthesize): With the amplitude $\hat{A}_k$ and noise estimates $\hat{\sigma}^2$ in hand, we now generate a synthetic signal. First, define the ground truth $\mathbf{t}$ time series as:

$$t_n = \sum_{k=1}^{K} \hat{A}_k \cos\left(2\pi k \frac{n}{f_s}\right), \tag{4.15}$$

and then color $\mathbf{t}$ with white noise to define synthetic signal $\mathbf{s}$:

$$s_n = t_n + \varepsilon_n, \tag{4.16}$$

where $\varepsilon_n \sim \mathcal{N}\left(0, \hat{\sigma}^2\right)$ is generated using, for instance, the Box-Muller transform [198], and K is the maximum monitored low frequency (we discuss our choice of K in the evaluation section). The synthetic signal length must be sufficiently long to capture variability due to noise. Given that a 1 Hz sinusoid passes through all velocities twice in one second, and by the widely used rule of thumb that thirty samples is a sufficiently large sample size, we construct 15 second[2] synthetic signals to

[2]This 15s duration is likely much longer than is actually necessary, but Pitch Pipe is fast as we later show. So we were not compelled to find a lower bound.

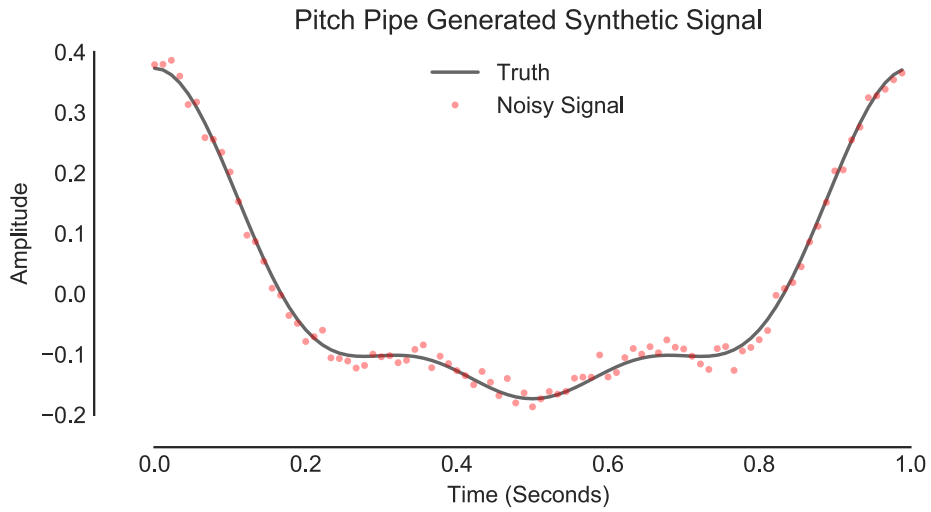


Figure 4.3: Part of the synthetic signal generated by Pitch Pipe after monitoring four seconds of HTC Vive controller input (see Figure 12.1). Although the shape is different, important characteristics of the input are preserved, including signal amplitude and noise.

ensure we have thirty samples at each velocity. Figure 4.2 (center) shows an example 4s synthetic signal generated for the HTC Vive controller data from Figure 12.1.

Step 3 (Optimize): Now that we have known ground truth $\mathbf{t}$ and synthetic signal $\mathbf{s}$, we need only to select an objective function of the form:

$$d(\mathbf{t}, \mathbf{y}) \rightarrow \mathbb{R}, \tag{4.17}$$

where d is an application-specific measure of dissimilarity between ground truth $\mathbf{t}$ and signal $\mathbf{y}$ (filtered $\mathbf{s}$). An optimizer then minimizes d by searching through the filter's parameter space. As a wrapper method, Pitch Pipe is not designed for a specific filter or objective function, but various options are possible and explored in our evaluation.

With respect to optimization, numerous techniques exist from which one may choose. However,

because we evaluate single parameter filters over objective functions that are sufficiently unimodel, we opt to use a golden-section search (GSS) [198] for most filters. Specifically, we first generate synthetic data per Equations 4.15 and 4.16, initialize the optimizer, and then perform one GSS iteration per frame until the solution converges. Once GSS completes, we generate a new sequence and repeat the optimization process, continuing as such, ad infinitum. The only exception is that we use a simple iterative linear search for the MA filter (Equation 4.18).

4.3 Evaluation Strategy

We evaluate Pitch Pipe over five unique input devices, four low-pass filters, and two objective functions. While establishing ground truth from noisy data is usually subjective and difficult, high quality Vive HTC controller input [174], with minor additional preparation, is suitable for large scale testing. Therefore, using the Vive high activity dataset described below, we conduct an omnibus test over varying noise levels and sampling rates to understand Pitch Pipe's capabilities. Thereafter, we conduct tracking tests on mouse and Tobii Pro eye tracker data, and gesture recognition tests on Kinect and Leap Motion data. In most cases, we will see that Pitch Pipe achieves near optimal performance.

4.3.1 The Datasets

Here we describe five of our six datasets, as mouse data is described later for readability.

Eye Tracking: We used an HTC Vive with Tobii Pro VR integration in order to collect eye-tracking data; we collected the Point of Regard along the X and Y axes over time. Nineteen participants were asked to wear the HMD and perform a simple visual attention task, by looking at stimuli as they appeared in a random position in their field of view. After viewing each stimulus, the user

was asked to look "straight forward". This process was performed 24 times for each participant, and data collection took approximately 5 minutes per user.

Low Activity (LA): JK2017 [235] comprises Kinect and Leap Motion continuous session data, collected from twenty unique participants per device for the purpose of custom gesture recognition testing. Each participant first provided two training samples per class, where there are 14 Kinect and 8 Leap Motion gesture classes. Thereafter, while playing a game of Simon Says, each participant performed every gesture three times in a randomized order. We refer to JK2017 as the low activity (LA) datasets because of long idle periods between gesticulations.

High Activity (HA): Inspired by JK2017 and for our own gesture recognition explorations, we collected a new dataset that combines gestures with direct manipulation and puppeteering-like interactions. From ten participants on Kinect and another ten on Vive, we first collected five training samples per gesture class, of which there are 17 full-body and 11 upper body gestures per device, respectively. Each participant then played a Follow the Leader (FTL) style game, where a participant follows an avatar's movement as closely as possible until a command appears. At this time he or she performs the requested gesture and immediately returns to following the leader. Consequently, each gesture, randomly performed three times, is interlaced with direct manipulation and puppeteer style motion. We refer to our FTL data as the high activity (HA) datasets because participants are continuously moving with high intensity.

4.3.2 The Filters

We used Pitch Pipe to tune four popular, widely adopted low-pass filters. Each technique approaches filtering from a different perspective and therefore focuses on a unique aspect of the common problem. This makes our selection appropriate for an omnibus test. In what follows, time series **x** is the input signal, and time series **y** is the filter's smoothed response.

MA: First, the moving average (MA) filter is a finite impulse response (FIR) approach that is optimal for removing white noise while preserving a sharp step response, which makes MA a common first choice [225]:

$$y_n = \frac{1}{M} \sum_{i=0}^{M-1} x_{n-i} \tag{4.18}$$

EMA: The exponential moving average (EMA) is an infinite impulse response (IIR) filter that uses weighted averaging, assigning less and less importance to samples by age [225]. This also immensely popular filter is defined recursively as:

$$y_n = \alpha x_n + (1-\alpha) y_{n-1}. \tag{4.19}$$

DEMA: Brown's double exponential moving average (DEMA), often used in forecasting, exploits trend information through the following relationship:

$$s_n^1 = \alpha x_n + (1-\alpha) s_{n-1}^1 \tag{4.20}$$

$$s_n^2 = \alpha s_n^1 + (1-\alpha) s_{n-1}^2 \tag{4.21}$$

$$y_n = 2s_n^1 - s_t^2. \tag{4.22}$$

This filter gained popularity after LaViola [136] demonstrated DEMA is on par with Kalman filtering for predictive tracking, despite being significantly faster and less complex.

Kalman: The Kalman filter is an advanced filtering technique that uses an internal process model to estimate system state, but then corrects prediction errors as data arrives. Its mathematics are omitted for space, but we implement the same position/velocity model used by LaViola [136]. Two parameters are required: one for the measurement noise covariance matrix and a second to scale the process covariance matrix. When tuning this filter, we use Pitch Pipe's noise estimate for the former and then optimize the scale parameter.

4.3.3 The Objective Functions

We explore two objective functions that serve unique application requirements. First, to tune for tracking, one will likely wish to minimize measurement error between the ground truth and sampled signal, i.e., to minimize the root mean square error (RMSE):

$$d(\mathbf{t},\mathbf{y}) = \sqrt{\frac{\sum_{n=0}^{N-1}(t_n - y_n)^2}{N}}. \tag{4.23}$$

where t_n and y_n are the ground truth and filtered samples, respectively. In the context of Pitch Pipe, these will be synthetic signals.

Second, in pattern recognition, one may favor smoothness over accuracy because common features used in 3D gesture recognition, such as the total angle traversed and path length [40, 235, 236], may scale undesirably with noise. Similarly, one measure used to classify time series that utilizes path length is the complexity invariant distance (CID) [15], a part of which is incorporated in Jackknife [235], the recognizer we use in our evaluation. This motivates an objective function based on path complexity:

$$d(\mathbf{t},\mathbf{y}) = \left| \left(\sum_{i=1}^{N-1} |t_i - t_{i-1}| \right) - \left(\sum_{i=1}^{N-1} |y_i - y_{i-1}| \right) \right|. \tag{4.24}$$

This objective function essentially measures the path length difference between the ground truth and filtered signal. All that remains now is to explain signal quality.

4.3.4 Signal Quality

The signal-to-noise ratio (SNR) in decibels is given by:

$$SNR_{dB} = 10\log_{10}\left(\frac{\sigma_s^2}{\sigma_n^2}\right), \tag{4.25}$$

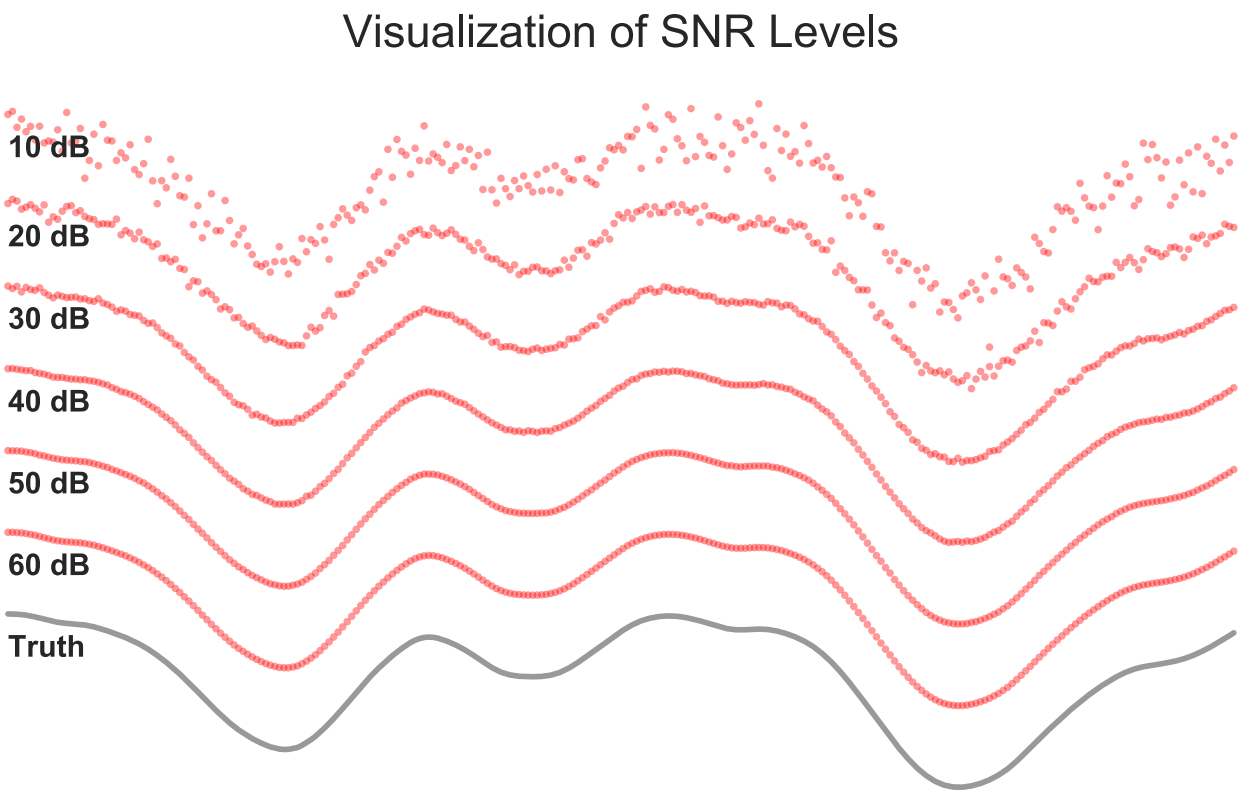


Figure 4.4: Visualization of how varying SNR levels affect signal quality. At 10 dB, we can still detect significant changes in amplitude despite heavy disturbance, though minor details are lost. By 50 dB, noise is practically imperceptible.

where σ_s^2 and σ_n^2 are the signal and noise variances, respectively. We illustrate varying SNR levels in Figure 4.4. Low quality signals in the 10—30 dB range are common, e.g. [181], and is where low-pass filtering may be especially beneficial. At 50 dB, where noise is hardly perceptible, filtering is likely unnecessary.

4.4 Omnibus Test

We performed an omnibus test using Vive HTC HA because of its superior signal quality [174], which we further improved with a zero phase, high order low-pass filter to establish a reliable,

non-subjective, yet sufficiently complex ground truth dataset. Thereafter, we used cubic spline interpolation to temporally resample signals to a given target sample rate and added Gaussian white noise to achieve a given SNR level per Equation 4.25. The purpose of this test is to evaluate Pitch Pipe over varying signal qualities and sample target rates. Each controller has three position coordinates, so that over ten users, there are $3 \times 2 \times 10 = 60$ samples per treatment.

4.4.1 Noise Estimation

For Pitch Pipe to be a useful solution, estimates must quickly and accurately converge. While low frequency amplitudes are measured directly as the signal arrives without issue, white noise is a low power stochastic process potentially masked by spectral leakage. To understand the limits of our noise estimator, we vary the sample rate f_s from 20 to 90 Hz and SNR from 5 to 60 dB, both in steps of five. This range enables us to see relevant trends. Overall percentage error results are reported in Figure 4.5 and convergence is shown in Figure 4.6.

We see that within the relevant noise level range, i.e., below 50 dB, estimation error is sufficiently low. As signal quality improves, the noise floor is harder to estimate and error increases. We also see that as the sample rate f_s increases, estimates improve, which is expected given that sample size per unit time also increases. With respect to the convergence speed, given a 40 dB signal, percentage error drops to 58% ($\sigma = 51\%$) after two seconds and then further drops to 16%($\sigma = 12\%$) after a half minute. At 20 dB, we see 83% ($\sigma = 96\%$) and 15% ($\sigma = 10\%$) for the same time periods. Both estimates continue to improve over time at a less dramatic rate. Informal preliminary testing on pure synthetic data suggests that these results lead to satisfactory performance.

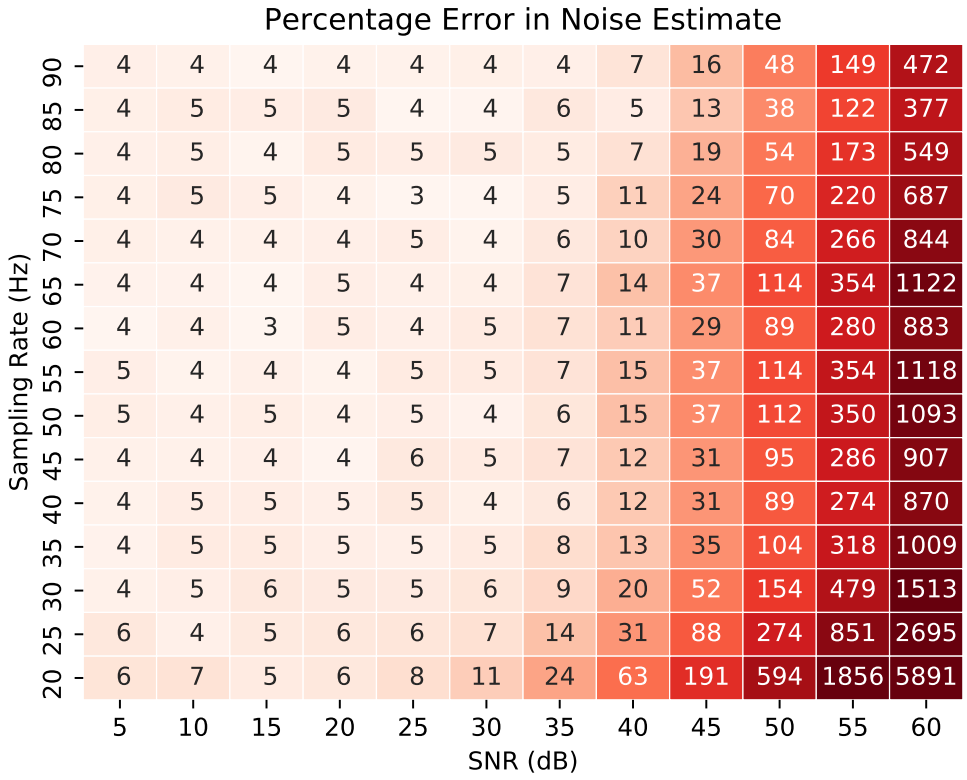


Figure 4.5: Heatmap of noise estimate percentage errors over varying sampling rate and SNR levels.

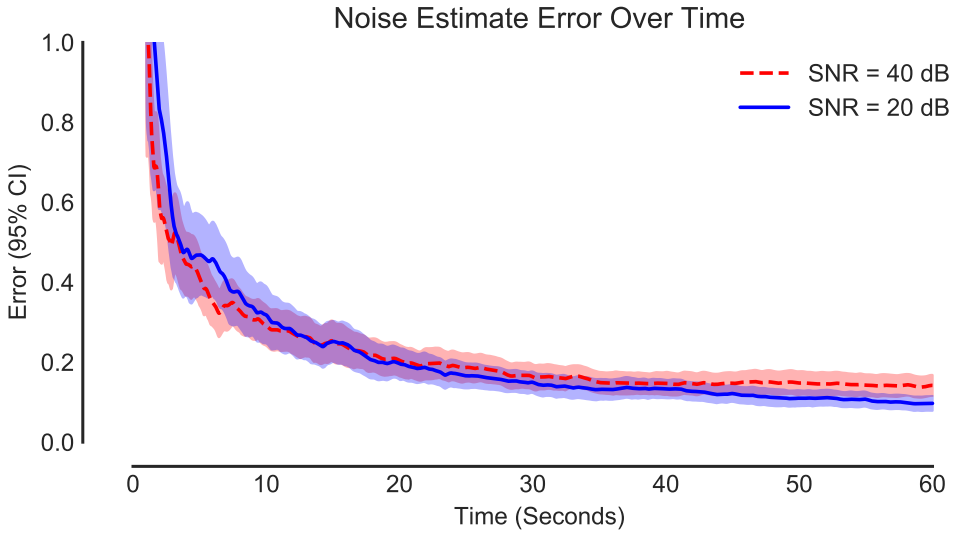


Figure 4.6: Noise estimate convergence over time

4.4.2 Auto-tuning

To understand how well Pitch Pipe auto-tunes a given filter, we first had to establish a baseline. We did so using standard practice—namely, we performed a grid search over the parameter space to find those settings that minimized the overall error for a given objective function and signal. This approach yields an optimally tuned filter possessing an unfair advantage of having seen the future, and which pits Pitch Pipe against difficult competition.

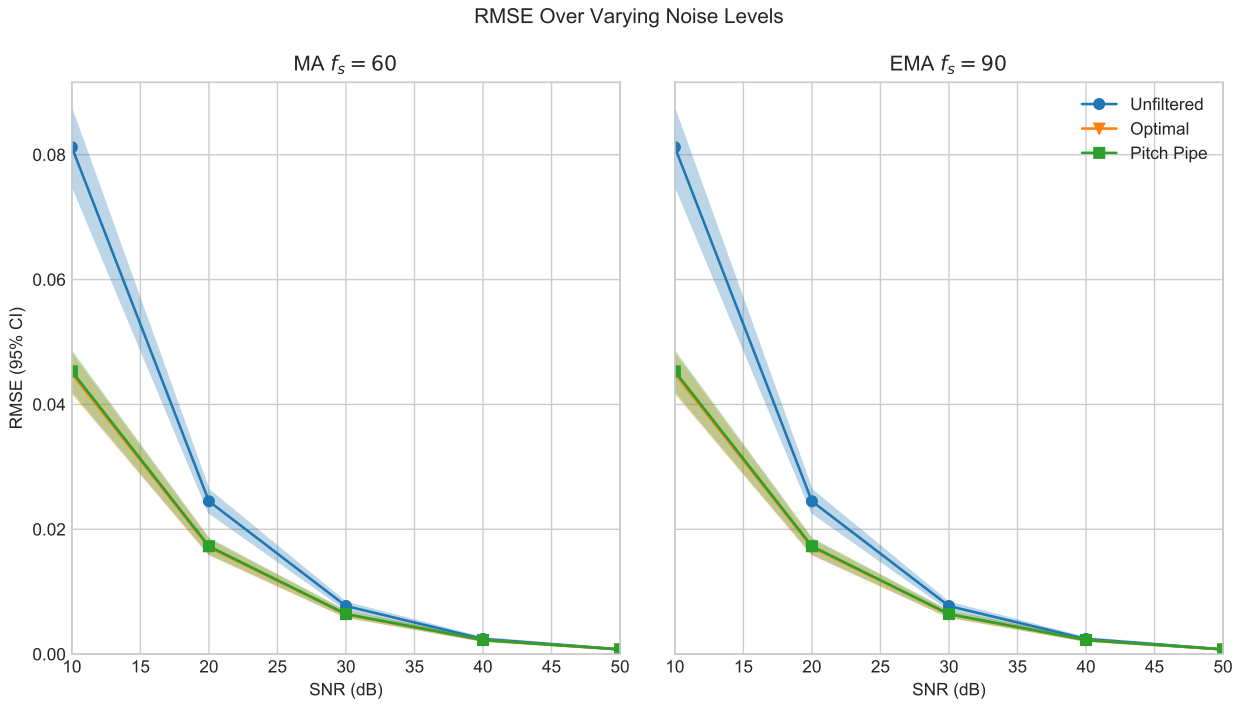
For logistics reasons, we reduced our test lattice relative to the noise estimate evaluation. Sample rates f_s now include 30, 60, and 90 Hz. SNR levels range from 10 to 50 dB over steps of 10. We also tested both objective functions over all low-pass filters. In aggregate, there were 10 participants $\times$ 6 components $\times$ 5 SNR levels $\times$ 3 sample rates $\times$ 4 filters $\times$ 2 objective functions $\times$ 2 methods = 14 400 tests. Because the HA datasets comprise coarse full body motions, we set the maximum tracked low frequency sinusoid to $K = 3$ (Equation 4.16) per [79, 227, 276]. Finally, we allow a five second warm up period before we begin to track errors.

All results are captured in Figures 4.7 and 4.8. We report percentage improvement relative to the unfiltered signal:

$$\text{Improvement} = \frac{\mathbf{x}_{rmse} - \mathbf{y}_{rmse}}{\mathbf{x}_{rmse}} \times 100\%, \tag{4.26}$$

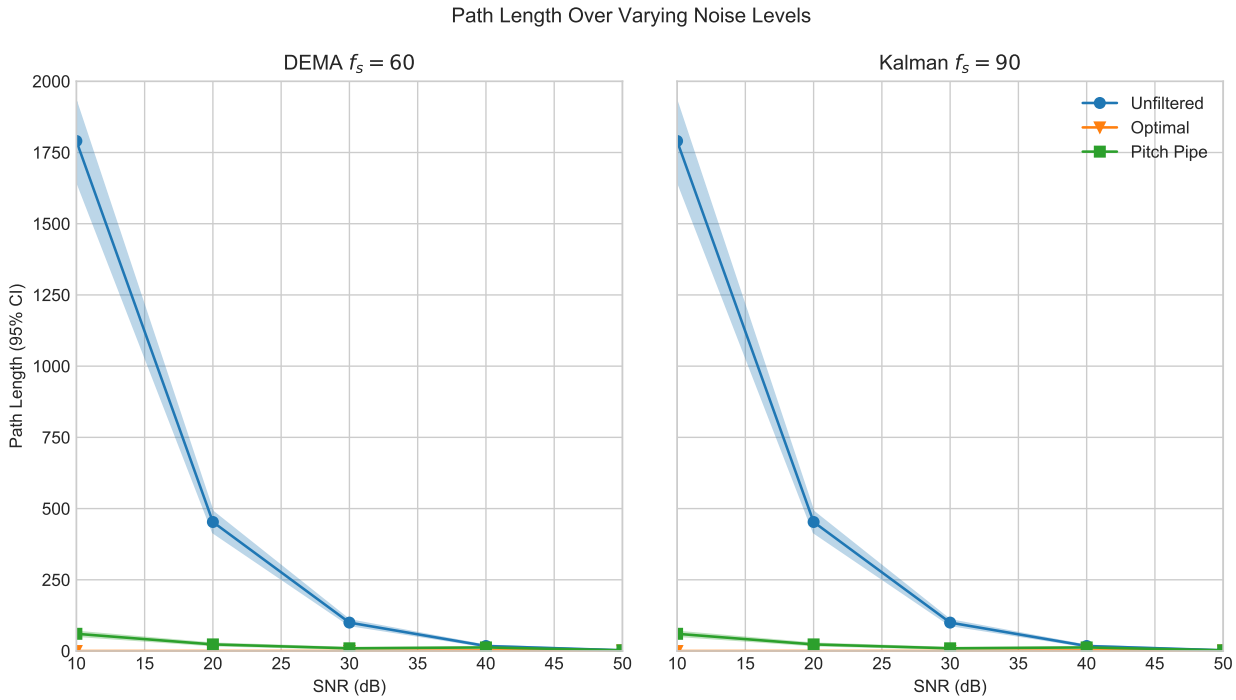
where $\mathbf{x}_{rmse}$ is the unfiltered signal's RMSE, and $\mathbf{y}_{rmse}$ is the filtered result. We similarly calculate path length errors.

We first observe that all Pitch Pipe RMSE improvements are within two percentage points of the optimum, and most are within one. For example, Pitch Pipe tuned MA reduces RMSE by 25% at $f_s = 60$ and 20 dB, which is close to the optimally tune MA result: 26%. This reason is also why one can barely distinguish Pitch Pipe from optimal in the accompanying graphs, regardless of sample rate, signal quality, or filter. In the best case, we see a 51% RMSE reduction at 10



f_s	Filter	10 dB		20 dB		30 dB		40 dB		50 dB	
		PP	(Opt)	PP	(Opt)	PP	(Opt)	PP	(Opt)	PP	(Opt)
90	MA	51	51	32	33	15	15	0	0	0	0
	EMA	51	51	31	32	13	13	3	3	0	0
	DEMA	51	51	37	38	24	25	13	13	5	5
	Kalman	51	51	39	39	27	27	16	16	7	7
60	MA	45	45	25	26	4	5	0	0	0	0
	EMA	45	45	25	25	8	9	1	1	0	0
	DEMA	44	45	29	30	17	17	7	8	2	2
	Kalman	44	45	31	32	19	20	9	10	3	3
30	MA	34	35	16	16	0	0	0	0	0	0
	EMA	34	35	14	14	3	3	0	0	0	0
	DEMA	32	33	17	17	7	7	2	2	0	0
	Kalman	31	32	18	19	8	9	3	3	0	1

Figure 4.7: Top: RMSE percentage reduction for four Pitch Pipe (PP) tuned filters and their off-line, optimally (Opt) tuned counterparts across varying sample rates f_s and SNR levels. An equal value indicates that the Pitch Pipe tuned filter peformed as well as the optimally tuned filter. Bottom: Exact RMSE values for select conditions (lower is better).



f_s	Filter	10 dB		20 dB		30 dB		40 dB		50 dB	
		PP	(Opt)	PP	(Opt)	PP	(Opt)	PP	(Opt)	PP	(Opt)
90	MA	98	98	96	100	93	100	85	100	67	101
	EMA	98	100	96	100	92	100	81	100	53	100
	DEMA	98	100	97	100	95	100	55	95	11	50
	Kalman	98	100	97	100	81	87	59	66	20	38
60	MA	96	100	94	100	89	100	74	100	25	100
	EMA	97	100	94	100	86	100	67	100	32	100
	DEMA	97	100	95	100	91	100	29	91	0	21
	Kalman	97	100	95	100	69	74	35	45	1	12
30	MA	91	100	86	100	70	101	15	101	0	31
	EMA	92	100	85	100	65	100	29	100	18	101
	DEMA	92	100	88	100	71	100	2	94	-23	8
	Kalman	92	100	89	100	32	41	0	6	-6	0

Figure 4.8: Top: Path length percentage error reduction for four Pitch Pipe (PP) tuned filters and their off-line, optimally (Opt) tuned counterparts across varying sample rates f_s and SNR levels. An equal value indicates that the Pitch Pipe tuned filter peformed as well as the optimally tuned filter. Bottom: Exact path length errors for select conditions (lower is better).

dB, which can occur with low quality sensors or when a user holds steady, e.g., while pointing. We also notice that performance improves as the sample rate f_s increases. Since many devices are able to sample well above 90 Hz, some into the thousands, we expect this trend to continue. Further, where percentage improvements trend towards zero, signal quality is high and tracking error vanishes, in which case, filtering is likely unnecessary.

At first glance, path length tuning appears to be less successful, yet upon further analysis, the results are actually compelling. We see that both the optimal solution and Pitch Pipe approach zero error, which is to say that the filtered and ground truth signals have near equivalent path lengths. For example, at $f_s = 90$ and 30 dB, the Pitch Pipe tuned EMA filter reduces path length error by 93%, whereas the optimally tuned filter achieves 100%. This result means that although the Pitch Pipe path length is a little longer than ground truth, such a large amount of error is removed that the result is still smooth.

4.5 Tracking on Other Devices

Although Pitch Pipe tuned filters achieved near optimal RMSE performance on Vive HTC HA data, we still wanted to verify that our systems worked well with other types of input. Therefore, we first performed a simple mouse tracking experiment, followed by a more extensive eye tracking evaluation. Further, we set the maximum monitored low frequency to $K = 4$ (Equation 4.16) since hand and eye movements tend to be faster than coarse full body gestures (though still generally well below 10 Hz) [79, 227, 276].

Mouse: Per prior filter testing methodology [35], we collected a 5.5 minute cursor position trace of one user interacting with their 2560 x 1600 (227 ppi) resolution display. We then down sampled

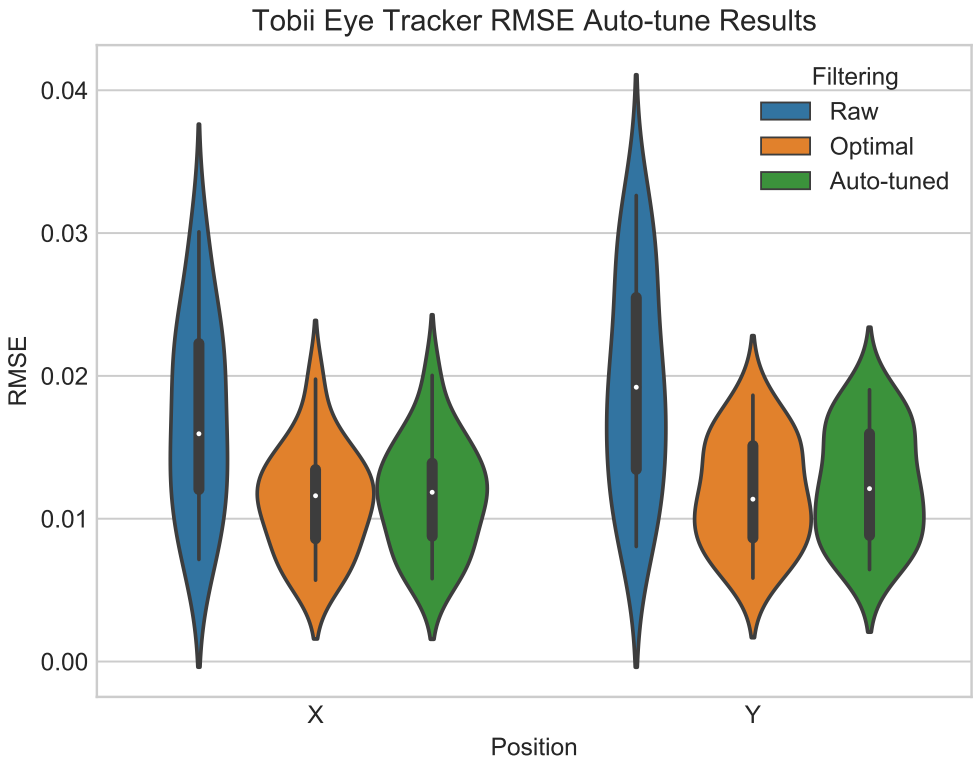


Figure 4.9: Violin plot of the Tobii eye tracker X and Y gaze position error over raw (unfiltered) data and Brown's DEMA filtered data, using both optimal and auto-tuned settings.

the trace to 100 Hz and added white noise so that its SNR measured 30 dB[3] per Equation 4.25. This noise level results in $\sigma = .82$ mm and $\sigma = .57$ mm jitter in the horizontal and vertical directions, respectively. Thereafter, with Brown's DEMA filter, we found the optimal smoothing parameter that minimized RMSE and evaluated Pitch Pipe's ability to auto-tune.

Along the horizontal axis, true RMSE error measured 7.41 pixels and with the optimally tuned DEMA filter, this dropped to 5.32 pixels. Pitch Pipe tuned DEMA achieved near optimal results at 5.34 pixels. We saw similar performance along the vertical axis: 5.11, 3.67, and 3.67 pixels for

[3] An on-line, interactive 1€ filter [35] demo utilizes screen resolution to determine noise variance for a given SNR level; see http://cristal.univ-lille.fr/~casiez/1euro/InteractiveDemo/ Using their approach, our SNR is approximately 50 dB, the demo's default noise level.

the same measures. Taking a different view of the same data, in either direction, both Pitch Pipe and the optimal solution were able to remove approximately 28% of the embedded noise, which is in line with trends found in the omnibus test (Table 4.7).

Eye Gaze: Using the same Vive HTC omnibus data preparation procedure, we cleaned the Tobii Pro eye tracker dataset and again ran the DEMA filter. After confirming that the per component unfiltered, optimal, and Pitch Pipe tuned DEMA RMSE results were normally distributed via a series of Shapiro-Wilk tests, we conducted pairwise, Holm-Bonferroni [100] corrected related samples t-tests to detect error rate differences. Note there are three pairs per component and six tests total. The unfiltered RMSE $(\mu = 0.019, \sigma = 0.0074)$ was significantly higher than optimal $(\mu = 0.0117, \sigma = 0.0036)$ with a large effect size $t(18) = 7.70,\ p < .001,\ d = 1.76$. Similarly, unfiltered vs Pitch Pipe $(\mu = 0.0123, \sigma = 0.0038)$ RMSE was significantly different with a large effect size $t(18) = 7.72,\ p < .001,\ d = 1.77$. Pitch Pipe versus optimal results were also significant, with a large effect size, $t(18) = 7.72,\ p < .001,\ d = 0.97$. The x-component results were similarly significant with large effects; so we omit their results. Analysis of the optimal and averaged auto-tuned smoothing parameter α, across all subjects and components, shows a strong and significant positive correlation $(r(36) = 0.85, p < .001)$.

Based on a percentage reduction in RMSE from unfiltered to optimal, approximately 39% of white noise was removed from the y-component signal, whereas Pitch Pipe achieved a 36% reduction. Improvements on the x-component signal are similar: 34% and 32%, respectively. Therefore, despite statistically significant differences between results, Pitch Pipe still performs very well, even without having been able to see the future (as did the optimally tuned filter).

4.6 Gesture Recognition Tests

Filtering for gesture recognition is different from tracking in that absolute error may be less important than smoothness, depending on the underlying technology. In this vein, we next evaluated Pitch Pipe in combination with a state-of-the-art recognizer on datasets of varying complexity. Specifically, we used CDP [179] to efficiently segment gesture candidates from a continuous data stream, after which we passed the candidates to Jackknife [235] for classification. To highlight the importance of using an appropriate objective function, we tested both RMSE and path length; and as with our Vive HTC omnibus test, we also performed a grid search to find the optimal solution. Further, we tested MA, EMA, and DEMA filters on the Kinect high and low activity datasets, as well as the Leap Motion LA dataset. Vive HTC HA was excluded because of its high signal quality and we wanted to avoid over using any one dataset. The Kalman filter was excluded for logistics reasons and because the other three filters outperformed Kalman in the path length omnibus test.

As with Vive HTC, we set the maximum monitored low frequency to $K = 3$. In this test, however, all components are monitored individually and then averaged together to produce a single synthetic signal. Thereafter, the filter is tuned as usual.

For both the Kinect and Leap Motion LA datasets, per participant, we trained the recognizer for custom gesture recognition, ran the participant's session through the system, and recorded all recognition errors. Our approach was similar for the HA dataset, except there are five training samples per gesture class. So, instead, we played each session ten times, selecting only two random samples per gesture class on each iteration. Since we are concerned with accuracy, we choose to use the popular F_1 score:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{4.27}$$

where precision is the fraction of true positives over all true and false positive results, and recall is

Table 4.2: F_1-score results with no filtering, Pitch Pipe tuned for tracking (RMSE), Pitch Pipe tuned for gesture recognition (Path Length), and the optimal solution. Percentage improvements relative to no filtering are also shown.

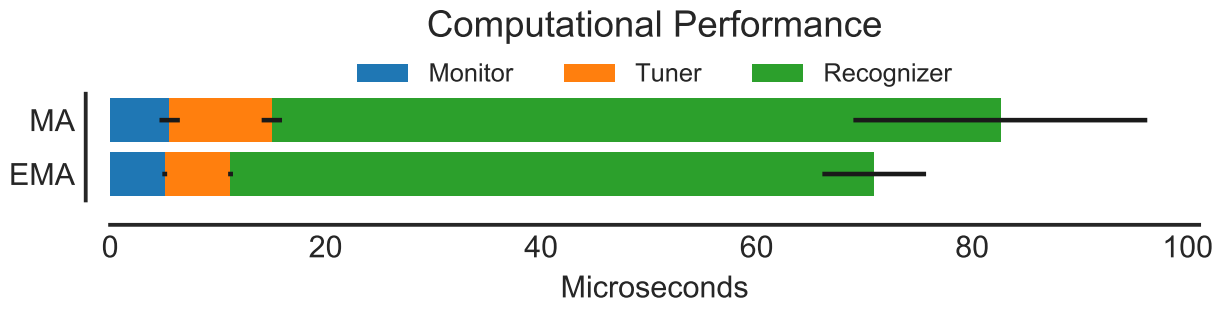
Dataset	Filter	No Filter		Auto-tuned RMSE			Auto-tuned Length			Optimal		
		μ	(σ)	μ	(σ)	%	μ	(σ)	%	μ	(σ)	%
Kinect HA	MA	89.9	(6.3)	90.1	(6.5)	2.5	92.8	(3.6)	29.5	92.8	(3.9)	28.9
	EMA	89.7	(6.4)	91.1	(5.0)	14.2	93.3	(3.5)	35.2	93.8	(3.4)	39.9
	DEMA	89.5	(6.3)	91.9	(4.7)	23.4	92.4	(3.4)	28.3	93.2	(3.9)	35.8
Kinect LA	MA	85.7	(15.8)	92.6	(8.3)	48.4	95.8	(6.1)	70.6	96.7	(4.8)	76.6
	EMA	85.7	(15.8)	92.5	(8.8)	47.3	96.1	(5.4)	72.5	96.7	(5.2)	76.8
	DEMA	85.7	(15.8)	92.4	(8.7)	46.8	96.5	(5.7)	75.8	96.7	(5.7)	77.2
Leap LA	MA	82.8	(11.0)	85.7	(8.2)	16.7	89.0	(9.9)	36.0	91.3	(9.4)	49.4
	EMA	82.8	(11.0)	86.3	(9.5)	20.2	90.5	(10.1)	44.4	91.4	(9.9)	49.8
	DEMA	82.8	(11.0)	85.7	(9.1)	16.6	91.2	(8.9)	48.5	91.7	(10.4)	51.8

the fraction of true positives over true positive and false negatives results.

We present all results in Table 4.2. One will first observe that in all cases, performance monotonically increases between no-filtering, RMSE, path length, and the optimal solution, except for in one case where path length just barely outperforms the optimal solution. We also observe that although tuning filters for tracking (RMSE) does improve accuracy over no filtering, path length results are much closer to optimal and able to achieve high accuracy (> 90% [137]). In the worst case (MA on Leap Motion), optimal performance reaches 91.3% accuracy, whereas Pitch Pipe only achieves 89%, a 2.3 point difference. This is still a 36% reduction in the overall error rate and an improvement over RMSE tuning. The remaining results are closer, and several are nearly in line with optimal. We further believe these results demonstrate the benefit of using Pitch Pipe.

4.7 Computational Performance

To verify computational efficiency, we recorded the average time per frame required to monitor all frequencies as well as auto-tune the MA and EMA filters, while running the Kinect HA dataset through our recognizer. We implemented our software in C++ and ran this test on a MacBook Pro with a 2.3 GHz Intel Core i5 processor having 8GB of 2133 MHz LPDDR3 memory. Results are summarized in Figure 4.10. Despite having to track 315 frequencies (twenty-one joints $\times$ three components per joint $\times$ five frequencies per component), the monitor only took 5.5 µs per frame, and auto-tuning took less than 10 µs in both cases, with MA being slower because of window-based averaging. Thus, the total additional overhead only increased latency by approximately 15 µs in the worst case.



Filter	Monitor		Tuner		Recognizer	
	μ	(σ)	μ	(σ)	μ	(σ)
MA	5.52	(0.94)	9.48	(1.93)	67.61	(13.64)
EMA	5.07	(0.22)	6.11	(0.34)	59.73	(4.81)

Figure 4.10: Average mircosecond time per frame required to monitor all frequencies, auto-tune the filter, and run the recognizer (exact values in table below graph).

4.8 Discussion

Filter tuning is a difficult task because there are numerous factors that affect performance, motivating the need for automatic parameter selection techniques. Pitch Pipe exploits the fact that human interactions involve only low frequency motion. That is, by monitoring only a few low frequency sinusoids, we gather sufficient user motion data, and from a couple of high frequency sinusoids, we reliably estimate noise. This minimal amount of information is surprisingly enough to generate representative synthetic data suitable for tuning a low-pass filter. Thus our approach enables Pitch Pipe to adapt to varying users, signal qualities, activity levels, and objectives, as shown through our diverse evaluation.

At its best, Pitch Pipe tuned filters reduced RMSE by 51%, yet optimally tuned filters could do no better. And throughout all Vive HTC tests, tracking error differences between Pitch Pipe and the optimal remained minor. In our mouse and eye tracking tests, optimal results were only marginally better than Pitch Pipe (36% vs 39% in the *worst* case).

Differences in path length error were more notable, though Pitch Pipe tuned filters still removed most noise from the signal, confirmed by observing that actual path lengths were near zero. In our gesture recognition test, we saw large jumps in accuracy between no filtering and RMSE Pitch Pipe tuned filtering. However, we saw even larger increases for path length tuned filtering. This confirms that one must consider the target application when choosing their filter parameters, and that automatic methods ought to support various options. Relative to optimal recognition, Pitch Pipe tuned filters performed almost as well—all results were within 2.5 points of the optimal with respect to F_1-score accuracy.

4.8.1 Limitations and Future Work

Perhaps Pitch Pipe's most significant limitation is that we do not include an automatic technique for selecting which low frequency sinusoids to monitor, that we instead rely on guidelines from prior research. We also assume input is heterogeneous rather than provide mechanics that adapt evolving use patterns. Finally, in this initial work, we focus on popular single parameter filters. In a preliminary investigation of multi-parameter filters, we found component-wise GSS easily falls into local extrema, and so we will have to consider alternative optimization techniques. These limitations are, of course, avenues for future work.

4.9 Conclusion

We have presented Pitch Pipe, a novel approach for automatic low-pass filter parameter selection that adapts to each unique situation. Through a simple, yet effective three step procedure, our general method auto-tunes low-pass filters according to workload, signal quality, and application variabilities. Further, through an extensive evaluation we have demonstrated that whether for tracking or gesture recognition, for mild or intense activities, for normal or low quality signals, and regardless of the input device, Pitch Pipe is able to achieve near optimal performance relative to filters that are pre-tuned over an entire sequence. We believe this shows that Pitch Pipe is a promising solution practitioners will find useful in a wide range of applications.

CHAPTER 5: PENNY PINCHER[1]

Work It. Make It. Do it. Makes us.
Harder. Better. Faster. Stronger.

More than. Hour. Our. Never.
Ever. After. Work is. Over.

Harder, Better, Faster, Stronger
Daft Punk

In this chapter we introduce Penny Pincher [231, 238] for rapid, custom gesture recognition over segmented input, which is our first of several techniques that make use of direction vectors as a primary representation of human motion. The virtue of using direction vectors is in its simplicity and speed, whereby using an inner product to measure dissimilarities between templates and queries allows for position and scale invariance using only multiplication and addition over a few data points. We first motivate the need for high speed custom gesture recognition, discuss our approach, and then present experimental results from several datasets as well as from an ecologically valid game designed to evaluate our recognizer. Ideas presented herein are enhanced and integrated into several components within The Dollar General pipeline.

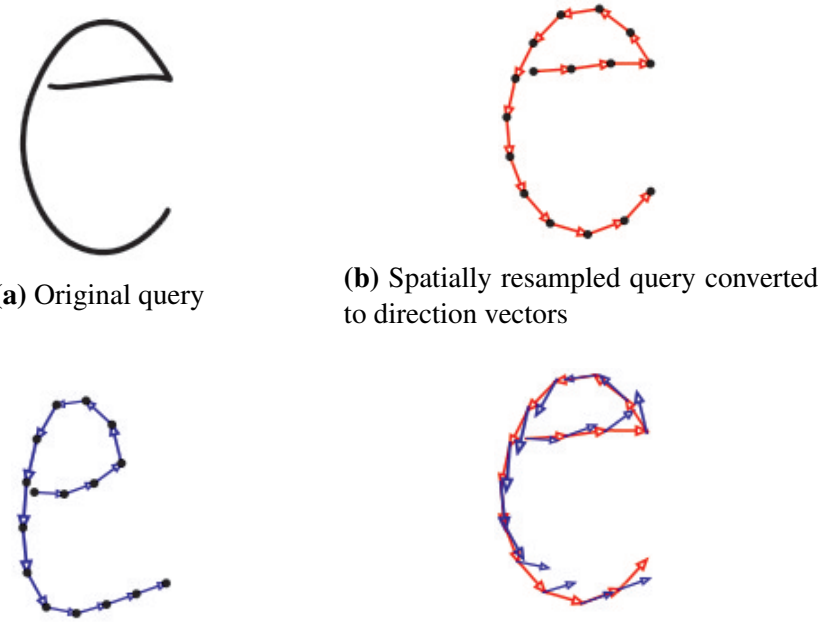
5.1 Introduction

To enhance our "need for speed" discussion from Chapter 2, suppose that a recognizer is part of a game. It must then share computational resources with several other components, some of which

[1]This chapter contains previously published material adapted from the following article: Taranta II, Eugene M., and Joseph J. LaViola Jr. "Penny pincher: a blazing fast, highly accurate $-family recognizer." Proceedings of the 41st Graphics Interface Conference. 2015. See Appendix C for associated copyright information.

include the renderer, animator, physics simulator, event and message handlers, network handlers, behavior scripts, and AI [87]. For this reason, a software architect may decide, for example, to allocate only 50 microseconds to the gesture recognition task on receipt of new input, thereby ensuring that the user interface does not disrupt other ongoing game processes. The architect's constraint poses two issues. First, with such a tight budget, one cannot use certain computationally complex techniques because they are simply too slow. Second, this constraint limits the number of templates a recognizer can evaluate during classification, even though more is better. Another concern is that recognition latency plays a critical role in user perception and interface design. Gray and Boeham-Davis [86] have shown that even seemingly negligible latencies (on the order of milliseconds) can influence how users interact with software. To address these concerns we designed Penny Pincher to be a streamlined custom gesture recognizer. This $-family extension is designed to do the absolute minimum amount of work possible when matching query 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 competitors when operating under a time constraint.

Whereas most other recognizers use Euclidean distance to measure queries relative to templates, we measure dissimilarities by comparing between-point vectors, see Figure 12.1. As is common, we spatially resample query strokes to a constant number of points; however, breaking from the traditional approach, we do not scale, rotate, or translate the input. Further, after resampling the input, we use only addition and multiplication operations—we avoid calls to computationally expensive geometric functions, such as *acos* or *sqrt*, to boost performance.



(a) Original query

(b) Spatially resampled query converted to direction vectors

(c) Template from database

(d) Comparison between the query and template

Figure 5.1: A query gesture (a) is resampled into a set of equidistance points along the trajectory and converted into a set of direction vectors (b). Given a template gesture (c) we compare corresponding direction vectors to find the best match (d). In this case, 'e' is the best match because it minimizes the sum of the angles between the corresponding vectors, compared to all other templates.

5.2 Penny Pincher

An input device samples a finite time series of points, which defines a gesture:

$$X = (\boldsymbol{x}_i \mid i = 1 \ldots X_N), \tag{5.1}$$

where $|X| = X_N$ is the number of samples in the series, i indexes time, and each datum $\boldsymbol{x}_i \in \mathbb{R}^2$. Thereafter, each gesture is spatially resampled to a set of N points so that the distance between each point is equidistance along the trajectory's path. This resolves variations in time and sampling rate,

but leaves us still to deal with scale and position. We handle the latter by converting each gesture into a set of between-point direction vectors as follows:

$$\vec{X} = (\vec{\boldsymbol{x}}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i \mid i = 1 \ldots X_N - 1)\,. \tag{5.2}$$

Notice that there is one less direction vector than there are points in the gesture. Further, because all vectors share zero as their origin, position is no longer unique per gesture.

Given a converted query $\vec{Q}$ and template $\vec{T}$, we can measure their dissimilarity f using a simple normalized inner product calculation:

$$f(\vec{Q}, \vec{T}) = -\sum_{i=1}^{N-1} \frac{<\vec{\boldsymbol{q}}_i, \vec{\boldsymbol{t}}_i>}{\|\vec{\boldsymbol{q}}_i\| \left\|\vec{\boldsymbol{t}}_i\right\|}. \tag{5.3}$$

With f, a perfect score is $1 - N$ because the normalized dot product is unity for identical vectors, and the result is negated to ensure the function measures their dissimilarity. Finally, we can infer the gesture class of query Q by measuring its dissimilarity against each template stored in the application's database, selecting that template which yields the least difference, and assuming Q belongs to the same class:

$$\mathbb{T} = \left\{ \vec{T}_t \;\middle|\; t = 1 \ldots T_N \right\} \tag{5.4}$$

$$g_i = G\left(\underset{T_j \in \mathbb{T}}{\operatorname{argmin}} f\left(\vec{Q}, \vec{T}_j\right) \right), \tag{5.5}$$

where $g_i = G(T_t)$ specifies template T_t's gesture class ($g_i \in \mathbb{G}$), and $|\mathbb{T}| = T_N$ is the number of templates.

This already simple calculation can be simplified further. During training, let the components of the between-point vectors be normalized so that each $\left\|\vec{\boldsymbol{t}}_i\right\| = 1$, thus eliminating one normalization

factor in Equation 5.3. Next, as a simplifying assumption, say that the length of each between-point vector component is equal (each $\|\vec{\boldsymbol{q}}_i\| = \left\|\vec{\boldsymbol{q}}_{i+1}\right\|$) because gestures are resampled into $N-1$ equidistance arc lengths, an assumption we evaluate in Section 5.3.2. This allows us to treat $\|q_i\|^{-1}$ as a constant to be factored out. Further, note that because this factor is the same for all evaluated templates in Equation 5.5—it scales each dot product result identically—we can eliminated the factor without affecting the final result, thus simplifying our dissimilarity measure to just:

$$f(\vec{Q},\vec{T}) = -\sum_{i=1}^{N-1} \vec{\boldsymbol{q}}_i \cdot \vec{\boldsymbol{t}}_i. \tag{5.6}$$

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 5.3.4. The simplicity of our approach also means Penny Pincher is the easiest to understand and implement of all $-family style recognizers.

We note that Penny Pincher is not specifically a multistroke gesture recognizer. However, we leverage the same technique used by $N [8], 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.

5.2.1 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. Query gestures are always resampled.
2. The resampled query gesture is always compared against every template in $\mathbb{T}$.
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 query 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 $t \in \mathbb{T}$, which is $\Omega(N \times |\mathbb{T}|)$. Therefore, at best, a template based recognizer as described above is bounded by:

$$\Omega(2R + N + N \times |\mathbb{T}|). \tag{5.7}$$

To the best of our knowledge, Penny Pincher is the first $-family style recognizer to achieve this lower bound. Note that the complexity of several other recognizers are provided by Vatavu *et al.* [250], however, these do not consider the cost of resampling. Of course Ω 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 shown in our evaluation. Penny Pincher, on the other hand, relies on straightforward dot product calculations without having to utilize external or built-in math libraries.

5.3 Evaluation

We evaluate Penny Pincher over three different experiments. All tests were performed on a MacBook Pro with a 2.4 GHz Intel Core i7 processor having 8 GiB of 1333 MHz DDR3 memory and a 760 GB mechanical SATA disk. In our first test, we check the validity of our assumption that all between-point direction vector lengths within a single gesture are approximately equal. 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.

5.3.1 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 work.

$1-GDS. The $1 recognizer dataset [267] 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 [4] 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.* [257] 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 [9] comprises 16 gestures having between one and three strokes. Samples were collected from 20 participants using either a stylus or touch on a tablet PC at three different speeds (slow, medium, and fast). In aggregate there are 9600 samples.

UJI. The UJIpenchars2 Database [147] is a multistroke gesture dataset comprising lowercase and uppercase letters, digits, characters with diacritics, and punctuation marks making an aggregate of 11640 samples over 97 gesture classes. Data was collected from 60 writers each providing 2 samples of each class.

5.3.2 Verification of Between-Point Direction Vector Distribution

The simplifying assumption used in Equation 5.6 supposes that each between-point direction vector is of equal length. In reality, the assumption is inaccurate because 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, we examine the empirical probability density distribution generated with the

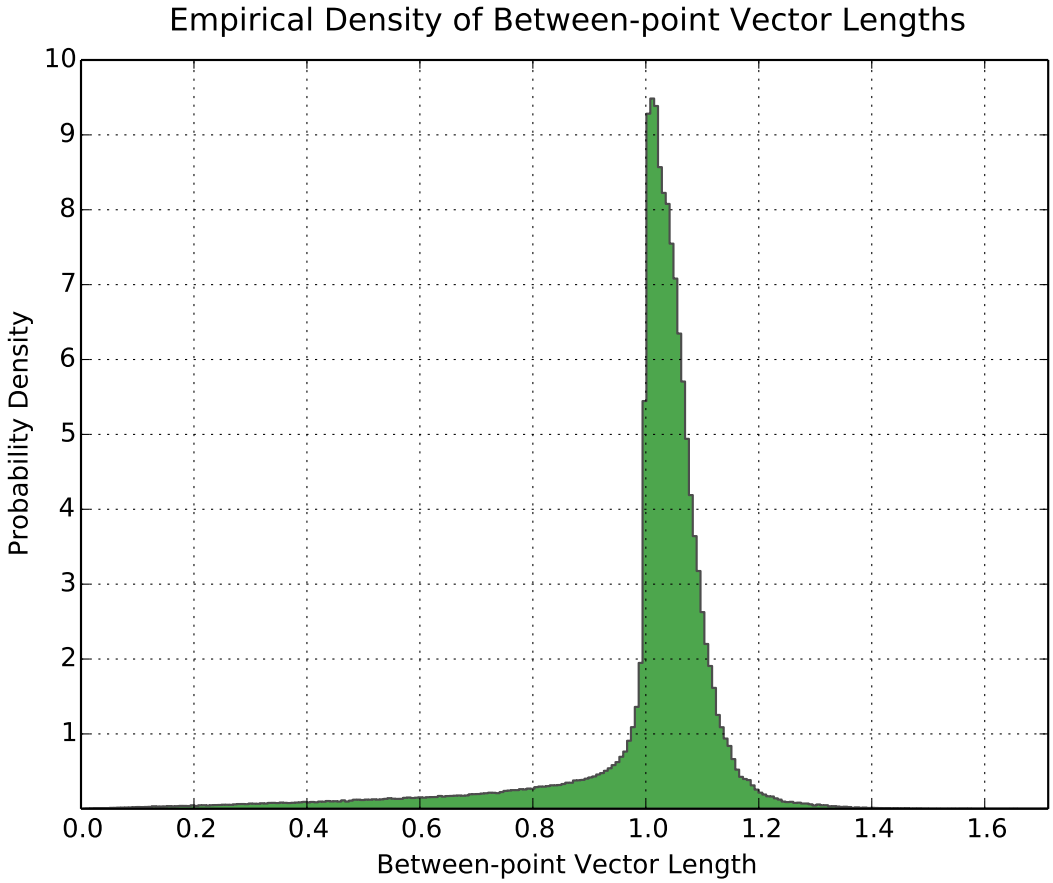


Figure 5.2: The empirical probability density of the distribution of between-point direction vector lengths. For each resampled gesture in $1-GDS, SIGN, EDS 1, and EDS 2, we normalize each vector by the mean length of all vectors within the gesture. All results are combined to create this overall distribution of relative lengths. It can be seen that the majority of the density is centered near 1.0, which indicates that the majority of vectors are of equal length within each sample.

relative between-point direction vector lengths of each sample across all six datasets ($1-GDS, SIGN, EDS 1, EDS 2, MMG, and UJI). For each individual sample, the mean between-point direction vector length is computed and all vectors within the same sample are normalized by the mean. Thereafter, the set of all normalized vectors from all sets are combined to build the empirical probability density distribution shown in Figure 5.2. If all between-point vectors were truly of equal length, the full density would fall on 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 the distribution 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%). 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, 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.

5.3.3 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 recognizer was set according to that given (or suggested) in each recognizer's paper respectively: 1¢, Protractor, and Penny Pincher resample to $N = 16$ points whereas \$N, \$N-Protractor, and \$1 resample to $N = 96$ 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 [250], 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

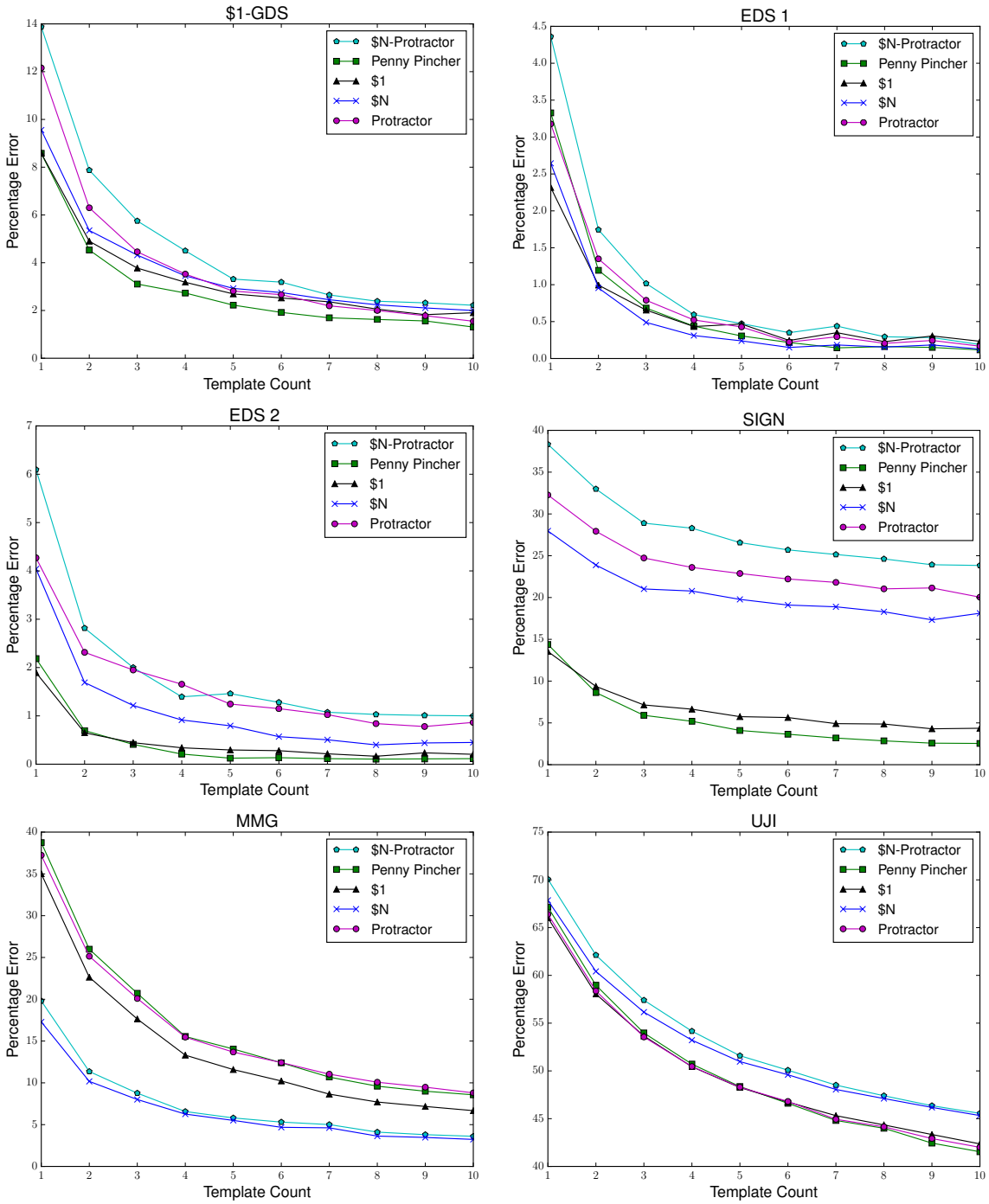


Figure 5.3: User independent error recognition test results for varying template counts. Each test (per recognizer and template count) was performed 1000 times. 1¢ was also tested, though because it exhibited high error rates with these datasets, it was removed for readability.

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 query gesture. Therefore, each iteration 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 its accuracy). All of the standard test results can be found in Figure 5.3.

Penny Pincher performs well 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 7 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 $1 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 all strokes and directions. So although the template count is 10, for example, in reality $N is evaluating sig-

Table 5.1: Average time taken in nanoseconds per template to perform a 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.

Recognizer	**Time per Template** (ns)	
	μ	(σ)
$1^{¢}$	25	(4.5)
Penny Pincher	33	(6.2)
Protractor	174	(27.7)
$N-Protractor	2449	(650)

nificantly 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. Moving on, 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%

Across all 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 5.1 shows the 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 ($T \times G$). The result is the time it takes to compare one query gesture to one template gesture (though note that the cost of resampling the query 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 within a given time constraint as reported in the next subsection.

5.3.4 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. Our 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 10µs budget and allow the recognizer to train with as many templates as it can process under the given constraint. Using this protocol, we execute 1000 randomized tests. Thereafter, the budget is incremented by 10µs and we rerun the test. 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 will depend 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. Results are shown in Figure 5.4, where graphs are cropped to 100µs for readability, though tail results are reported in the discussion.

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 11µs. As the test continues, the dataset is exhausted with a 111.29µs 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 an 11µs budget. This improves to only 0.53%. Protractor starts with the worst accuracy, only being able to handle 1.05 TPG at 10µs.

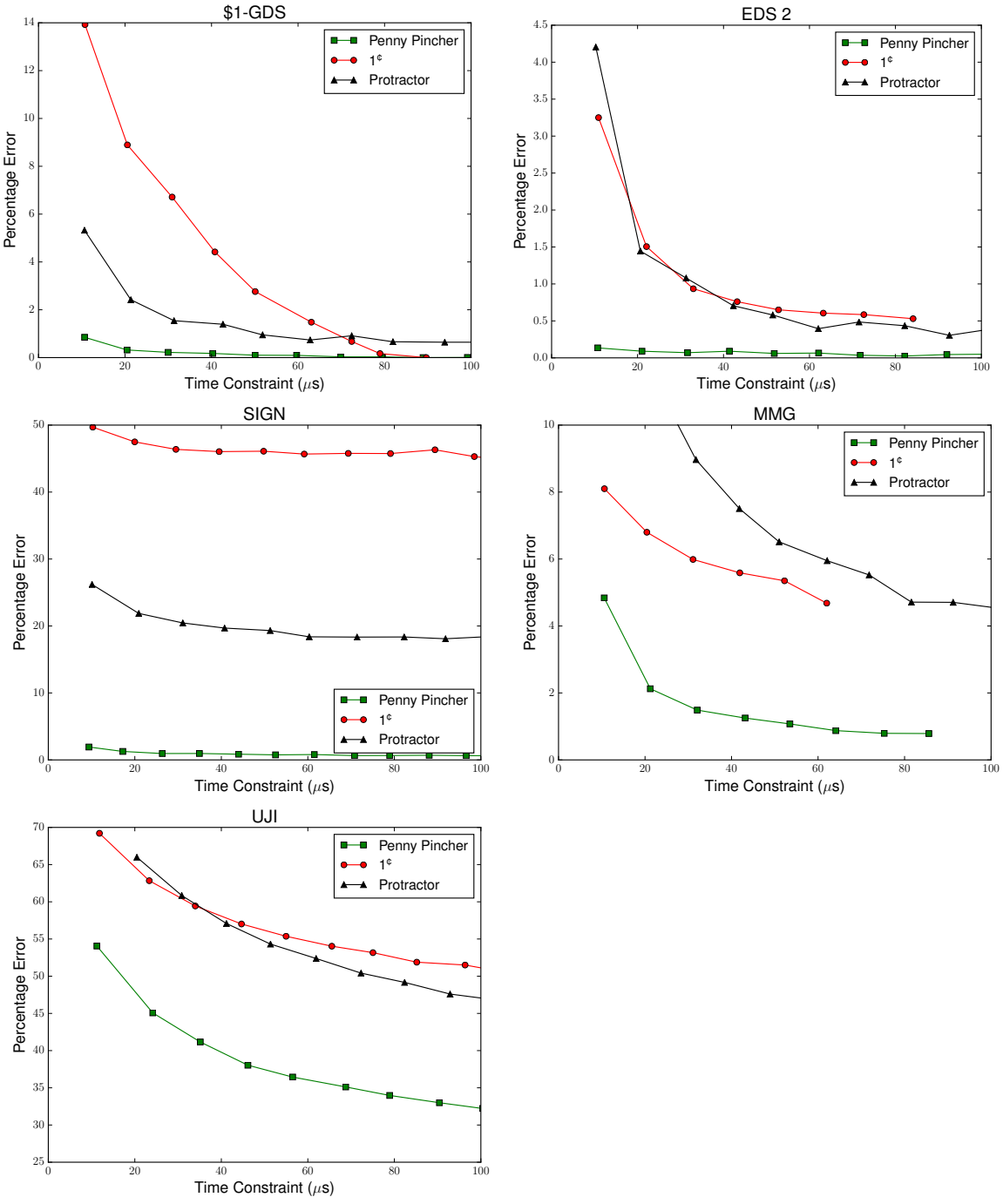


Figure 5.4: Mixed user error recognition test results under varying time constraints. Each test (per recognizer and budget) was performed 1000 times. Although the graphs are cropped to 100μs for readability, the tail results are reported in Section 5.3.4.

As expected, Protractor takes the longest to complete where the dataset is exhausted at 701.34µs, though the error rate drops to 0.1%. Across all budgets, Penny Pincher achieves a considerably lower error rate.

Of all our tests, 1¢ achieves its best results on 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 90µs. Penny Pincher also reaches 0% with 219.38 TPG at 89µs. However, its initial error is much better, 0.85% with 19.12 TPG at 11µs. Starting with 2.81 TPG, Protractor only achieves a 5.3% error and at its best, cannot reach 0%. Using 297.56 TPG over 766µs, Protractor reaches 0.2% error.

UJI again is the most challenging dataset. Even at 65.81 TPG and 135µs, 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 1152µs budget to do so. Penny Pincher has the best performance overall. With 3.06 TPG at 11µs, the initial error is 54%, which steadily drops to 28% by 183µs.

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 the 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 11µs can achieve a 4.8% error, which declines rapidly to 1.1% (42.19 TPG) at 21µs, and then to 0.79% at 86µs (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 526µs and 187 TPG at 62µs).

SIGN again is where we see the most dramatic difference in performance between Penny Pincher and the other recognizers. With a budget of 9µs and 14.29 TPG, the error is already 1.9%. This

Table 5.2: 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 5.4 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 MMG, the dataset is exhausted before Penny Pincher reachers 100µs, which is why there is no entry.

	20µs		**40**µs		**60**µs	
$1-GDS	87%	(0.32, 2.41)	87%	(0.18, 1.39)	87%	(0.09, 0.74)
EDS 2	94%	(0.09, 1.45)	87%	(0.09, 0.71)	84%	(0.07, 0.40)
SIGN	94%	(1.26, 21.88)	95%	(0.96, 19.69)	96%	(0.75, 18.38)
MMG	69%	(2.13, 6.80)	83%	(1.25, 7.51)	85%	(0.88, 5.95)
UJI	26%	(45.07, 60.84)	30%	(38.04, 54.32)	30%	(35.12, 50.42)

	80µs		**100**µs	
$1-GDS	95%	(0.03, 0.66)	99%	(0.01, 0.64)
EDS 2	94%	(0.03, 0.44)	87%	(0.05, 0.38)
SIGN	96%	(0.65, 18.36)	96%	(0.69, 18.40)
MMG	83%	(0.79, 4.71)	—	—
UJI	31%	(32.99, 47.60)	31%	(31.41, 45.7)

continues to improve until the end where the error drops to 0.54% (with 1936 TPG over 1213 µs). 1¢ struggles initially in comparison with a 50% error that only drops to 43% near the end with 747µs. Protractor is in the middle with an initial 26% error (2.41 TPG) which improves only to 14% with a 5720µs budget.

In Table 5.2 we provide a summary of select results from the budget test. For each dataset-budget pair, we compare Penny Pincher to the second best performing recognizer and report the percentage reduction in recognition error. The exact errors are shown next to each result. It can be seen in all cases that Penny Pincher significantly outperforms the other recognizers. Across 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%.

5.4 In-game Evaluation

There are a number of practical applications for pen and touch gesture recognition, though one of particular interest to us is in the electronic games domain. There are two reasons for this: use cases are limited only to the imagination of game designers, and gesture recognition in games is difficult, often falling below expectation. Baseline results from offline tests, for example, are often better than those results collected in a video game environment [38, 236], which in part may be due to the induced stress associated with playing a game itself. Therefore, to evaluate the effectiveness of Penny Pincher and other $-family recognizers in this difficult but practical situation, we developed the toy game *Lemarchand's Prototype* shown in Figure 5.5, which is loosely inspired by Lemarchand's box from Clive Barker's novella *The Hellbound Heart* [13].

In our bimodal game, a box and four wheels that are arranged symmetrically in the center of the screen are rotated using touch, and symbols are drawn with a stylus. The box has a window in one corner that exposes symbols etched into the wheel over which the window is oriented. Upon selecting the desired wheel by rotating the box, the wheel itself is then rotated by touch to select from its different symbols. Those wheels not selected rotate automatically to ensure the player is able to see all symbols in play. Once the desired symbol is inside of the box's window, a player can draw the symbol (gesture) anywhere on the screen. If a unistroke symbol is recognized as the selected symbol within the window, and if there is an enemy on the screen matching that symbol, then the enemy is banished to another realm. Enemies are arms that grope their way towards the box, and once close enough, shoot lightning (or "magic") at the box in order to wreak havoc. As mentioned, each arm has a symbol on its hand which matches one of the symbols on one of the four wheels. The associated symbol on the wheel is highlighted lightly once the corresponding arm is visible on the screen to assist the player with locating matches. When a gesture is recognized and the enemy is banished, a new symbol appears on the wheel; this continues until the game ends. At

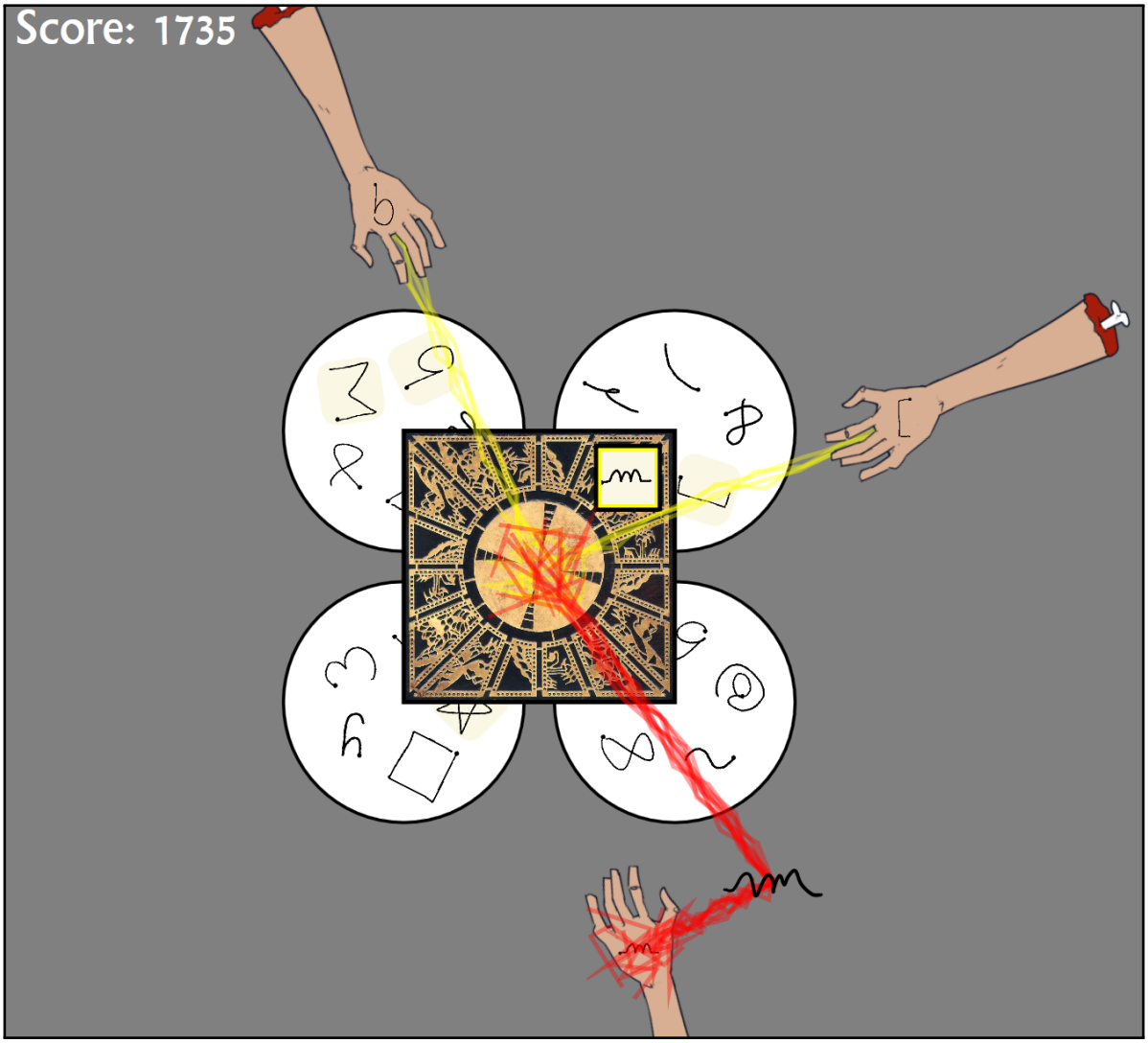


Figure 5.5: Screenshot of Lemarchand's Prototype, a game designed for pen-based gesture input. The box and wheels are rotated with touch. If the symbol in the box's window matches the symbol on the hand of an enemy, then the player can draw the symbol to banish it. Enemies move faster and are generated faster as the game progresses.

the start of a round, a player has approximately 10–14 seconds to banish an enemy before it reaches the cube and is able to attack. Similarly, there is a sufficient delay between when new enemies are created so that the game pace is leisurely. As the game progresses, enemies are generated faster and move faster (approximately 3—6 seconds to reach the box) so that not all arms can be defeated

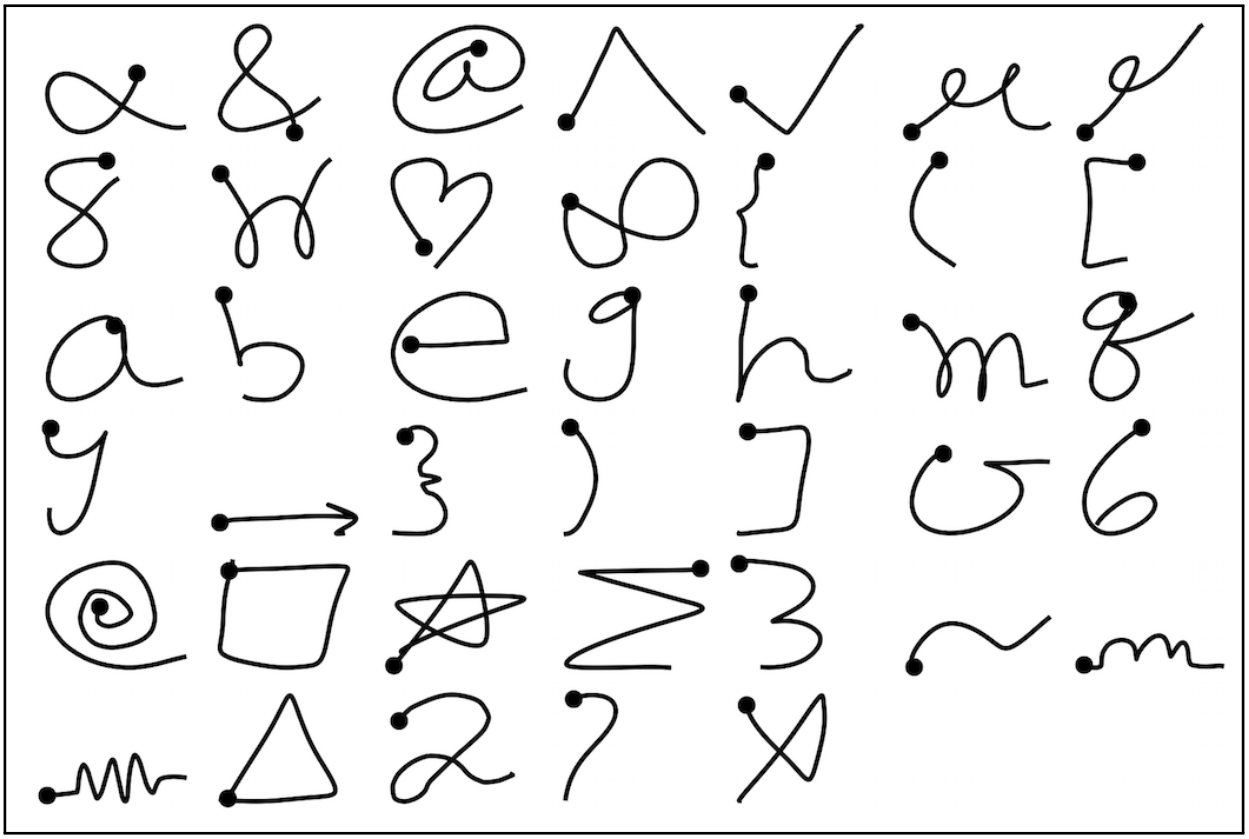


Figure 5.6: The 40 gestures used in Lemarchand's Prototype. These are randomly selected samples collected from participants during training.

before they begin to attack. At this time, players are forced to react as quickly as possible in order to defend the box before too much damage is taken. By the end, there are usually six to twelve enemies attacking at once.

The Lemarchand's Prototype gesture set comprises the 40 unistroke symbols shown in Figure 5.6, which were inspired by different sources. The star and right arrow variants, unistroke 'x', caret, check, pigtail variant, and curly and square brackets were derived from \$1-GDS [267], which we extended with the rounded brackets. The '3', '6', '8', 'a', 'g', 'm', and triangle variant were derived from EDS 1 [257]. The 'α', rectangle (square), heart, curly, '?', and spiral variant are due to EDS 2 [257]. The '2', '@', 'b', 'e', 'h', 'q', and 'y' gestures come from UJI [147]. We also

included the division gesture (second row, second column in figure) from the Graffiti [1] gesture set. Finally, from no particular source, we also added the '&', '∞', 'Σ', 'σ', '∼', transistor, and register symbols.

5.4.1 Experimental Design

Since our evaluation involved a large number of direction sensitive unistroke gestures, we were only concerned with recognizers designed for such. $1^{¢}$ generally does not perform well when gestures are symmetric or rotation variant; Protractor is considered to be a $1 enhancement[2] because of its speedup and similar (sometimes better) accuracy; $N as well as $N-Protractor are extensions designed to also handle multistroke gestures; and $P is a general purpose, stroke order and direction invariant recognizer. Following the cheat sheet provided by Vatavu *et al.* [250], we evaluated our recognizer alongside Protractor because of its support for stroke direction sensitivity, low coding complexity, and low algorithmic complexity. For the in-game Protractor recognizer, we integrated the reference source code available on the $1 project website[2]. The standard way of evaluating $-family recognizers is to increment the template count from 1 template per gesture to the maximum available (usually 9 or 10). In practice, however, a user is unlikely to provide more than one example of each gesture [143], or perhaps at most, two to three. Further, it would be quite time consuming to have a participant play the game 10 times, once for each count of templates per gesture from one to ten, and performance would likely deteriorate as participants become lethargic with repeated game play. As a compromise, we settled on a 3-level within-subjects design for each recognizer and between-subjects design between recognizers. That is, participants were asked to play the game three times utilizing one recognizer, each iteration with a different number of templates per gesture: 1, 3, and 10. Each recognizer was evaluated with 12 participants (24

[2]See the $1 unistroke recognizer project website: https://depts.washington.edu/aimgroup/proj/dollar/

total) recruited from the University of Central Florida, and each participant was compensated $10 for approximately 65 minutes of time. Of the participants that utilized Protractor, 10 were male (2 female), 10 were right handed (2 left), all but 1 had prior experience with pen and touch interfaces (2 had a lot of experience), and the ages ranged from 18—24 (average 21.3). Of the participants that utilized Penny Pincher, 7 were male (5 female), 9 were right handed (3 left), all but 2 had prior experience with pen and touch interfaces (none had a lot of experience), and the ages ranged from 14—34 (average 21.6).

5.4.2 Procedure

Each participant was first acquainted with the experiment and then asked to provide 10 samples of each gesture. Our data collection apparatus allows for five samples to be collected at a time, each in a different area on the display. Above each location, an example of the required gesture was shown. Further, the order of the 400 samples (forty gestures times ten) were randomly permuted. After training data was collected, the participant filled out a pre-questionnaire to collect demographic information. Next the participants played a tutorial round of the game where all of the mechanics were introduced, though the recognizers were not active at this time.

After the practice round was completed, a participant played the game three times: with 1, 3, and 10 templates per gesture loaded. However, the order was randomly permuted, and since there were three levels, there were six possible permutations. Having twelve participants, we ensured that each permutation was tested twice. During a single iteration of the game, the order of the enemy gestures were randomly selected, and each symbol appeared three times so that once the experiment was completed, each gesture was attempted at least nine times. However, a single instance of a gesture can be attempted at most three times. After three unsuccessful attempts, the associated enemy is banished regardless. Note that the second and third attempts were included to

allow the participant additional practice in case of bad form, though we only consider first attempts when calculating accuracy.

5.4.3 Baseline Results

Using the training data only, we first estimated baseline performance—the highest accuracy possible when there are no additional stress factors to consider. Specifically, we examined two scenarios: user dependent accuracy and user independent accuracy. A user dependent scenario may represent a situation where a user provides templates for gestures that only he or she will write in the game. For example, this may occur when custom gestures are mapped to certain commands within a game, or when user dependent recognition is expected to be an important part of the game play. User independent recognition, on the other hand, may be important in a variety of different scenarios such as:

- A player borrows her friend's mobile device to try out a game trained with her friend's gestures.
- A multiplayer game involves cooperative gesture creation and manipulation such that ad hoc gestures are shared amongst the players.
- A hobbyist developer or student collects training data from his or her family and friends that are then integrated into a prototype game.
- An application does not support data collection or training and deploys with a user independent gesture recognition interface.

Our baseline tests follow the same protocol as the standard tests where template and candidate gestures are randomly selected from the participant's training data. Results are shown Figure 5.7.

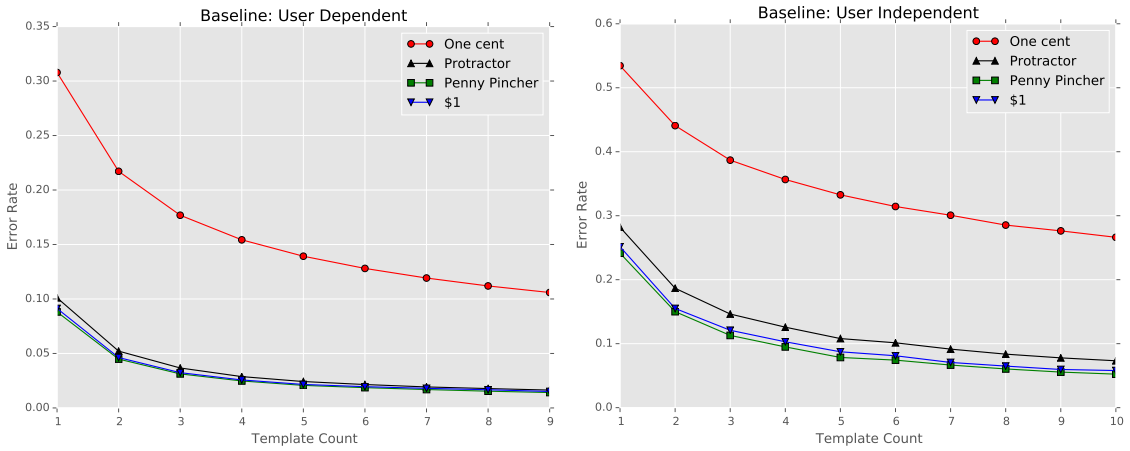


Figure 5.7: Baseline accuracy for user dependent (left) and user independent (right) test case scenarios based on training data as queries and templates. Note that in the user dependent case, the template count only goes up to nine. This limit is because one sample per gesture is saved as a candidate gesture to be classified in each test, and there are only ten samples collected per gesture.

Although in-game data (as discussed in the next subsection) was collected and evaluated for only two recognizers, we ran this evaluation over all of the unistroke gesture recognizers. In the user dependent scenario, the $1 and Penny Pincher recognizers perform almost identically and are the best performing recognizers. At one, two, and three templates per gesture, Penny Pincher achieves 8.8%, 4.5%, and 3.1% recognition error rates, whereas $1 is at 9.1%, 4.6%, and 3.2%; and by ten templates per gestures, they fall to 1.4% and 1.5% respectively. Penny Pincher, therefore, sees only a very slight advantage over $1. Protractor is also close, starting out a 10.1%, 5.2%, and 3.7%, and drops to 1.6% for the same template counts, and 1¢ is the worst performer overall with this particular dataset.

The relationship between the recognizers in user independent baseline results are similar; although as expected, all recognition error rates are worse due to differences in style and handwriting between participants. Again, $1 and Penny Pincher achieve close results, with Penny Pincher being slightly better. At one, two, and three templates per gesture, Penny Pincher achieves a 24.1%,

15.0%, and 11.3% recognition error rate, and by ten templates per gesture, this drops to 5.2%. $1's recognition error rate is not as low, but still reaches 25.2%, 15.5%, and 12.1%, and 5.8% for the same template counts. The third best recognizer is Protractor, which falls slightly behind the others at 28.2%, 18.7%, and 14.6% for one, two, and three templates, and 7.3% by ten templates per gesture.

These initial baseline results suggest that Penny Pincher should do as well or even slightly better than Protractor in the game. Further, although $1 was not explicitly evaluated in the game, Penny Pincher should perform almost identically.

5.4.4 In-game Results

The in-game recognition error rates for Protractor and Penny Pincher are shown in Table 5.3. Considering baseline results, we noticed that Protractor and Penny Pincher were further apart in recognition error rates than expected. Upon further inspection, we found that the reference source[2] scales strokes for Protractor as well as $1. However, this is not necessary with our dataset, and in fact, performance is much better without scaling. Therefore, we also include results for the in-game data using Protractor without scaling, and further discussions refer to this latter result. It can be seen that the Penny Pincher recognition error rates are superior for all levels, where the error rate is decreased by 16.1%, 21.8%, and 10.5% for one, three, and ten templates per gesture respectively.

For the remaining tests in this subsection, we used samples collected during game play to evaluate user dependent and user independent scenarios for Penny Pincher, $1, Protractor and 1¢. We utilize this configuration to ensure that all recognizers are evaluated with the same distribution of online samples. The experimental setup is similar to the standard test (Section 5.3.3) except that the candidate gestures are any of those gestures that were drawn during game play by any participant.

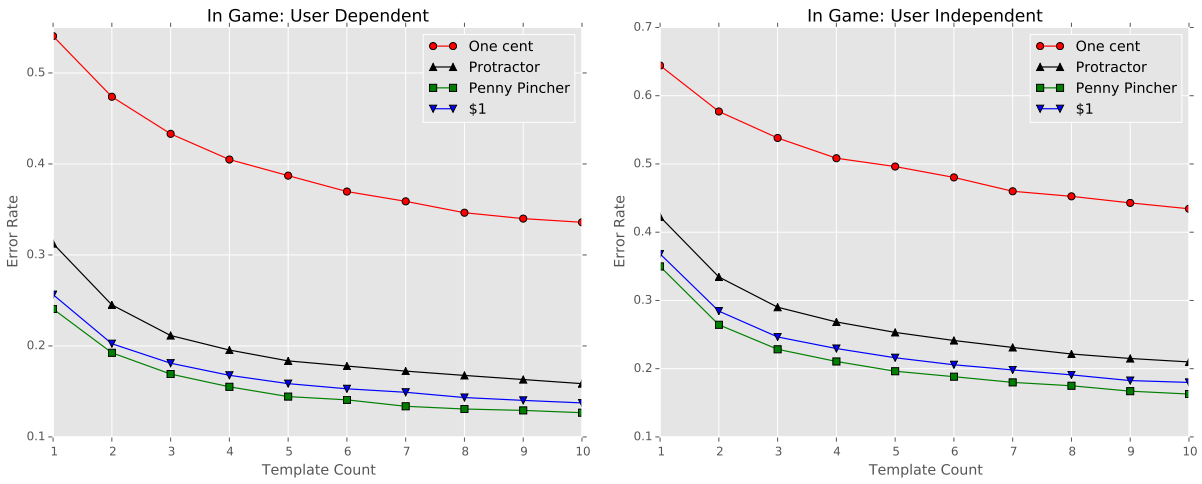


Figure 5.8: Recognition error rate of user dependent (left) and user independent (right) test case scenarios using the first in-game gesture attempts as the candidate gestures and training data as the template gestures.

Table 5.3: Summary of baseline and in-game recognition error rates for Protractor and Penny Pincher for 1, 3, and 10 templates per gesture. Because the in-game protractor implementation scales strokes, we also report the error rates without scaling, which in our tests are much lower.

T	**Protractor**			**Penny Pincher**	
	Baseline	Game	Without Scaling	Baseline	Game
1	11.9%	34.4%	27.4%	8.8%	23.0%
3	4.6%	24.4%	22.4%	3.1%	17.5%
10	2.1%	19.0%	13.3%	1.4%	11.9%

Since there are 24 participants and 9 first attempts of each gestures, there are 216 candidates per gesture to choose from in each of the 1000 iterations. Further, recall that the template gestures are chosen at random from the training data in each iteration.

With the in-game data, the difference between Penny Pincher and $1 is more pronounced than in the baseline test, see Figure 5.8. With one, two, and three templates, Penny Pincher achieves recognition errors rates of 24.1%, 19.2%, and 16.9% respectively, which drops to 12.7% by ten

templates. $1 sees 25.6%, 20.2%, and 18.1%, which then drops to 13.7% for the same template counts respectively. This represents a 5.8%, 4.9%, and 6.6% reduction in error rate for the first three template counts, and a 7.3% reduction for ten templates. Protractor is slightly less accurate than $1: 27.9%, 22.2%, and 19.7%, which then drops to 14.6%. Compared to the baseline test results, the in-game accuracy of Penny Pincher dropped by 16% at one template per gesture and 11.6% at nine templates per gesture in the user dependent test. $1 accuracy also dropped by 18.2% and 12.6% for the same template counts. So, although in the user dependent baseline tests, Penny Pincher and $1 were close in their recognition errors rates, $1 had a larger overall drop with the in-game data. Protractor's drop was even more dramatic, 21.9% and 14.5%.

When looking at the user independent test results, it can be seen that the trends are very similar. Penny Pincher, again, is the best achiever in recognition errors rates: 35.0%, 26.4%, and 22.8% for one, two, and three templates per gesture, which drops to 16.3% by ten templates. This is followed by $1: 36.8%, 28.4%, and 24.7% for the lower template counts, and 18.0% for ten templates. Protractor is not too far behind at 40.1%, 31.6%, 27.3%, and 20.1% for the same four template counts. Between Penny Pincher and $1, our recognizer enjoys a 5.1%, 7.5%, and 8.3% reduction in error for up to three templates per gesture, and by ten, the reduction reaches 10.4%. It is worth noting that the margin between $1 and Penny Pincher is somewhat consistent in all of the charts (except the baseline user dependent test) as the template counts increase. This implies the percentage reduction in error rate continues to improve as the template count increases, even if the difference in accuracy is somewhat constant.

5.4.5 Misclassified Gestures

In a game where one has to quickly find a symbol, mechanically arrange components through touch, and then rapidly pen a gesture, mistakes are going to be made. This is supported by the fact

that our baseline results are dramatically different from the in-game results, as is often the case [38, 236]. However, it is also important to understand the cause of these errors. In a post analysis of first-attempt misclassifications, we visually examined each recorded sample that was classified incorrectly and further divided all errors into four categories: misfire, mistaken, malformed, and false negative. A **misfire** occurs when one accidentally strikes the pen against the interactive display or when one starts, but does not complete a gesture. One cause of this may be that the participant realized that he or she was about to pen an incorrect gesture. Switching between touch and pen may be another cause of misfires, and in a couple of cases, participants tried to rotate the box or a wheel with the stylus. To determine misfires, we assumed that the symbol framed by the selection window was the gesture attempted. It is possible that a participant performed a gesture without first selecting the correct symbol, and though we do not have a method to differentiate this situation, it does not seem apparent upon visual inspection of the collected gestures. A **mistaken** gesture occurs when one gesture is expected but a different gesture is drawn. A common example of this occurred when a participant should have drawn an '@' gesture, but he or she drew a spiral instead. These errors may have been caused by rushed game play, where the participant did not properly observe the expected gesture and mistook, by broad shape similarity, a different gesture. **Malformed** gestures occur when the stroke is readable—a human can recognize the symbol—but the form is wrong; one example of this is when a player draws an α symbol backwards, starting at the wrong end of the stroke. The $N and $P recognizers can handle this situation, but in our game, we do not allow stroke direction invariance. Finally, a **false negative** is simply a well formed gesture that is confused and misclassified as another.

Note that the distinction between malformed gestures and false negatives can be subjective. For instance, samples that are human readable but excessively sloppy or misshaped are classified as malformed, though there is no formal criteria. It is also important to understand that a malformed gesture can be correctly recognized, especially when the gesture exists in the "sloppiness space"

Table 5.4: Distribution of misclassification error types for first attempt gestures of in-game Protractor and Penny Pincher results. Note that there were 4320 first attempts per recognizer: 12 participants by 40 gestures by 9 instances.

Error Type	Protractor		Penny Pincher	
	Count	Percentage	Count	Percentage
Misfire	118	10.5%	121	15.8%
Mistaken	71	6.3%	56	7.3%
Malformed	517	45.8%	329	43.1%
False Negative	423	37.5%	258	33.8%
Total	1129	—	764	—

[85] where its form is sufficiently far from neighboring gestures such that its sloppiness does not cause a problem, which depends on the recognizer's internal representation of gestures, gesture set size, and distributions in the feature space.

The tabulated results are shown Table 5.4. Misfired and mistaken gestures were similar for both Protractor and Penny Pincher in raw numbers. This is not surprising given that participants were under the same level of stress and their interaction with the game was expected to be similar. On the other hand, we noticed that malformed gestures and false negatives were significantly higher with Protractor (with the in-game Protractor recognizer variant). However, the percentage distribution between malformed and false negative gestures is almost identical for both recognizers. As mentioned before, we expected participants (on average) to utilize both recognizers identically; therefore, this result is not surprising given that sloppiness is likely uniform, and because Protractor appears to be less accurate with this particular dataset, the raw number of misclassifications are accordingly higher without changing the distribution. The false negative and malformed gesture counts would likely have been lower with Protractor's scaling removed, as previously discussed. Despite a high number of misclassifications, especially with Protractor, users did not report being frustrated while playing the game, and most enjoyed the challenge.

5.5 Discussion

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 direction vectors. This immediately afforded us two benefits—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 direction 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 we 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. In one case, the dataset was a multistroke dataset having varying stroke orders and directions, which is why Penny Pincher did not perform as well as $N and $N-Protractor with only a few templates loaded.

Computational performance, however, was initially 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 60µs. When considering reductions in recognition error rates for 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 of all the $-family style 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 with an implementation in an interpreted language running on a mobile device, budgets would likely have to be much higher to attain the same levels of accuracy.

Our derivation of the recognizer assumes that the between-point direction vectors are of equal length. While this is not strictly true, the empirical probability distribution of relative 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, 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, through 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.

We were surprised to see Penny Pincher perform so well as compared to the other recognizers when testing involved lower template counts. However, the majority of datasets evaluated were collected in a relaxed stress-free lab environment. This made us curious about how well our recognizer could do in a more difficult setting, again using a small number of templates as often occurs in practice, which is why we developed Lemarchand's Prototype—to evaluate the effectiveness

of Penny Pincher in a stressful game setting, one in which users are likely to be less precise in their writing. As it turns out, Penny Pincher still performs remarkably well. Compared to $1 and Protractor, the average recognition error rates were lower with a 5.8% to 10.4% reduction in error rate. Even though in the user dependent test, Penny Pincher scored just slightly better than its closest competitor, the difference was more pronounced in our user independent tests. These results in combination with previous results suggest that Penny Pincher is a good general recognizer for small and large datasets, and that because of its simplicity, our recognizer will likely be preferred by many for its speed, quick implementation, and high accuracy.

5.5.1 Limitations and Future Work

Through our evaluations, we demonstrated the benefits of Penny Pincher, but it is also important to understand its various limitations. Since we compare between-point direction 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 may essentially have the same effect, and therefore, it may be unnecessary to derive the numerous permutations (some of which may never occur in practice) to achieve high accuracy. This is supported, in part, by the fact that we achieve over 99% accuracy in the budget test with the MMG dataset, and Penny Pincher also achieves high accuracy (relative to $N and $N-Protractor) with UJI.

The idea behind our work is that there are potentially hundreds of templates available per gesture 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 for a user interface. As we saw in the evaluation though, Penny Pincher provides high accuracy already with even a small number of templates. So over time, as the system is used, additional examples can be collected and the recognizer can be continuously retrained with new data, thereby improving accuracy without incurring a major performance penalty.

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. Finally, there are numerous ways in which games utilizing gesture recognition can be imagined. In order to further validate the effectiveness of Penny Pincher and other recognizers in a game environment, additional studies are warranted. One example includes a multi-player tabletop fighting game in which four or more players are interacting and as a result, numerous gestures are being produced at varying angles and orientations, which is a potentially very challenging scenario to handle.

5.6 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. Finally, with an online user evaluation of Penny Pincher and Protractor in a video game, we were able to demonstrate that Penny Pincher still yields higher accuracy. Utilizing the in-game data we collected, we also evaluated $1 and 1¢, varying template counts, as well as the user independent and user dependent use case scenarios. In all tests, Penny Pincher achieved the highest accuracy with our dataset. In the next chapter, we will look how we can apply Penny Pincer principles to 3D data.

CHAPTER 6: JACKKNIFE[1]

Work it harder. Do it faster.
More than ever, our work is never over.

Work it harder. Make it better.
Do it faster. Makes us stronger.

Harder, Better, Faster, Stronger
Daft Punk

Efforts to provide robust custom gesture recognition for input devices beyond pen and touch have been limited to special cases. Kratz and Rohs address accelerometer sensor data generally and Wiimote input specifically with $3 [128] and Protractor3D [129]. Liu *et al.* do the same with uWave [146], a dynamic time warping (DTW) based recognizer also designed for accelerometer input. Caputo *et al.*'s 3 cent recognizer [31] handles midair hand gestures. These example recognizers are input device specific, perhaps hinting at generality, but never taking the full step forward. This chapter presents Jackknife, the first device-agnostic custom gesture recognizer adhering to $-family principles. Jackknife [235] builds on and subsumes Penny Pincher by utilizing between-point direction vectors as its primary representation of human motion, but unlike its predecessor, Jackknife affords elastically in its matching and scoring scheme via DTW. Jackknife can optionally operate on raw data rather than vectors when scale and position are unimportant. However, because differences in accuracy between raw and vectorial data are minor, we default to the latter representation. In another aspect of this work, dissimilarity scores are further improved by accounting for information not immediately accessible to the underling measurement function. Correction factors (inspired by CIDDTW [15]) inflate the baseline DTW score by incorporating

[1]This chapter contains previously published material adapted from the following article: Taranta II, Eugene M., *et al.* "Jackknife: A reliable recognizer with few samples and many modalities." Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. 2017. See Appendix C for associated copyright information.

differences in geometric measures between the query and template under investigation, such as the difference in bounding box widths. We find that DTW over direction vectors when combined with correction factors provide for robust recognition, as we show through an extensive evaluation across several public datasets. For this reason, we employ Jackknife in our 3D gesture recognition pipeline.

6.1 Jackknife

For convenience, we repeat basic definitions stated in previous chapters. First, we define a gesture instance as a finite time series of samples collected from an input device:

$$X = (\boldsymbol{x}_i \mid i = 1 \ldots X_N), \tag{6.1}$$

where $|X| = X_N$ is the number of samples in the series, i indexes time, and each datum $\boldsymbol{x}_i \in \mathbb{R}^m$. Typical values of m for various modalities include $m = 2$ for pen or touch, $m = 21 \times 3$ for Kinect 2.0, and $m = 21 \times 3$ for Leap Motion. We may also represent a given gesture as a set of direction vectors through space as follows:

$$\vec{X} = (\vec{\boldsymbol{x}}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i \mid i = 1 \ldots X_N - 1). \tag{6.2}$$

This form has the advantage of being position invariant since all directions originate from the origin. To remove scale, we further normalize the vectors:

$$\hat{X} = \left(\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_{i+1} - \boldsymbol{x}_i}{\|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\|}. \;\middle|\; i = 1 \ldots X_N - 1 \right). \tag{6.3}$$

Further, we denote a query as Q and a template as T, where a template is a time series representa-

tion (model) of a specific gesture class. Jackknife employees 1-nearest neighbor classification—a given query is compared against all templates stored in a database, and the template whose measure against the query is least dissimilar infers the query's class. Formally, given a set of templates $\mathbb{T} = \{T_t \mid t = 1 \ldots T_N\}$ and query Q, the query's class is the gesture class of $T_i \in \mathbb{T}$ that minimizes:

$$T_t = \underset{T_i \in \mathbb{T}}{\operatorname{argmin}} \prod_{j=1}^{F} f_j(T_i, Q) \tag{6.4}$$

$$g_i = G(T_t), \tag{6.5}$$

where $g_i = G(T_t)$ specifies template T_t's gesture class ($g_i \in \mathbb{G}$), and $|\mathbb{T}| = T_N$ is the number of templates. Function f_1 is the underlying DTW measure of T against Q, and $f_{2 \leq i \leq F}$ are correction factors. Our approach is based on the complexity-invariant distance (CID) measure [15]. CID inflates a base distance measure, such as Euclidean distance or DTW, with a correction factor (CF) that is a measure of the complexity difference between a template and query. CF ranges from one to positive infinity such that time series similar in "complexity" score near one and the base distance measure remains relatively unchanged. Otherwise, the score is inflated. Our interpretation of CID is that a good CF is able to capture information about the dissimilarity of two time series for which the base distance measure is unable; however, in our view, the CF measure does not necessarily need to relate to notions of complexity. Before we further discuss CFs in detail, let us first understand DTW.

6.1.1 Dynamic Time Warping

Dynamic time warping (DTW) is an elastic matching dissimilarity measure that finds the optimal alignment between two sequences based on a given local cost function. Although squared Euclidean distance is the most common cost function, Jackknife utilizes the inner product of gesture

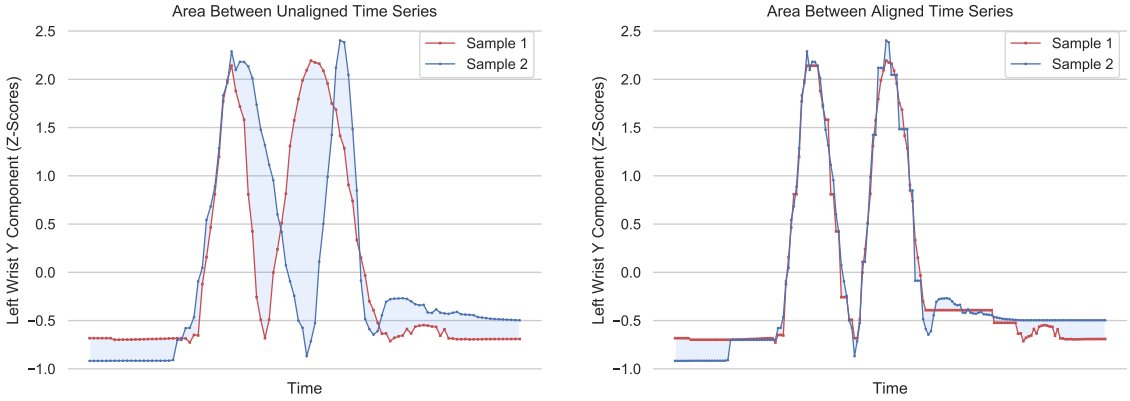


Figure 6.1: Visualization of two time series before and after DTW alignment. The shaded regions between them represent the amount of dissimilarity measured by Euclidean distance. Although the unaligned series are of a shorter duration, the area between them after alignment is significantly improved.

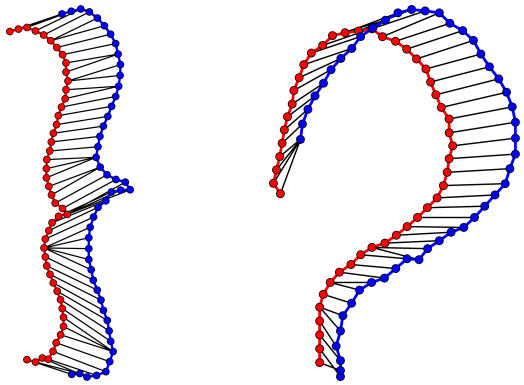


Figure 6.2: Visualization of the warping path found by DTW between two right curly braces (left) and two unistroke question marks (right), from the $1-GDS dataset [267].

path direction vectors, which provides translation and scale invariance. To understand how DTW works at a high level, consider Figure 7.1. We present two sequences produced by the same individual who twice in a row raises their arm up high. In the left graph, notice that the sequences are temporally out of sync because of differences in production speed. The shaded area between each sequence measures the squared Euclidean distance, *i.e.*, $\sum (x_i - y_i)^2$ between them, which reveals

major dissimilarities. Contrast this with DTW which is able to resolve temporal differences as shown in the right graph, where we see that DTW eliminates most of the shaded area. How does this work? Internally, DTW constructs an optimal *warping path* W that specifies how each element from query Q maps to template T. Warping path W is defined as:

$$W = \Big(\boldsymbol{w}_i = (a,b) \;\Big|\; 1 \leq a \leq |Q|, \quad 1 \leq b \leq |T| \Big), \tag{6.6}$$

and is used by DTW in the following way:

$$DTW(Q,T) = \sum_{i=1}^{|W|} d(\boldsymbol{q}_a, \boldsymbol{t}_b) \tag{6.7}$$

$$a = \boldsymbol{w}_{i_a}$$

$$b = \boldsymbol{w}_{i_b}$$

where $d(\boldsymbol{q}_a, \boldsymbol{t}_b)$ is a local cost function that measures the dissimilarity between query element $\boldsymbol{q}_a$ and template element $\boldsymbol{t}_b$.

Next we need to understand how DTW constructs the warping path. Let us suppose we have partial warping path $W[1:N]$ and the last element therein $\boldsymbol{w}_N = (i,\ j)$ maps query $\boldsymbol{q}_i$ to template $\boldsymbol{t}_j$. From here, we have three options for how we can proceed—we can set:

- $\boldsymbol{w}_{N+1} = (i,\ j+1)$ Repeat query $\boldsymbol{q}_i$ so that it is matched with template $\boldsymbol{t}_{j+1}$
- $\boldsymbol{w}_{N+1} = (i+1,\ j)$ Repeat template $\boldsymbol{t}_j$ so that it is matched with query $\boldsymbol{q}_{i+1}$
- $\boldsymbol{w}_{N+1} = (i+1,\ j+1)$ Repeat neither and match query $\boldsymbol{q}_{i+1}$ with template $\boldsymbol{t}_{j+1}$

It is by repeating elements in either sequence that DTW is able to align the sequences. Referring back to Figure 7.1 (right), one can see example short plateaus in the aligned curves where elements are repeated. Another illustration of two 2D warping paths are shown in Figure 6.2.

One might note that there are many possible warping paths that align two sequences, but DTW selects the optimal solution via an elegant dynamic programming approach. DTW constructs a $|Q| \times |T|$ matrix M, where each element (i, j) holds the optimal warping path score that aligns $Q[1:i]$ with $T[1:j]$. In other words, once two subsequences are aligned, element (i, j) is the measure between them. The matching decision described above can then be decided using the following recurrence relation:

$$M_{i,j} = d(\boldsymbol{q}_i, \boldsymbol{t}_j) + min \begin{cases} M_{i,j-1} & \text{Repeat query } \boldsymbol{q}_i \\ M_{i-1,j} & \text{Repeat template } \boldsymbol{t}_j \\ M_{i-1,j-1} & \text{Repeat neither,} \end{cases} \tag{6.8}$$

with boundary conditions:

$$M_{0,0} = 0 \tag{6.9}$$

$$M_{i,0} = \infty \tag{6.10}$$

$$M_{0,j} = \infty, \tag{6.11}$$

All that remains is to decide a local cost function.

6.1.1.1 Local Cost Function

As noted above, the local cost function $d(t_i, q_j)$ in Equation 7.5 most frequently used is the squared Euclidean distance over z-score normalized sequences[2]: $d(t_i, q_j) = (t_i - q_j)^2$. Though, we find in gesture recognition that this cost function is not always the best option. An alternative that has received less attention is the inner product of a feature vector measure [154], which can also be applied to gestures. Instead of extracting feature vectors from a time series, we utilize the gesture path direction vectors (see Equation 6.3). Since the inner product is a similarity measure in $[-1, 1]$ for unit vectors, we convert this to a dissimilarly as follows:

$$d(\hat{\boldsymbol{q}}_{\boldsymbol{j}}, \hat{\boldsymbol{t}}_{\boldsymbol{i}}) = 1 - < \hat{\boldsymbol{q}}_{\boldsymbol{j}}, \hat{\boldsymbol{t}}_{\boldsymbol{i}} > \tag{6.12}$$

where $1 \leq i < n$. This approach is inspired by Penny Pincher (see Chapter 5), which also uses the inner product of direction vectors, an approach that proved to be empirically faster than alternative unistroke recognizers while remaining competitive in accuracy. When used as a local cost function (as we later demonstrate) the inner product measure (IP) is often superior to the squared Euclidean distance measure (ED) in gesture recognition problems.

6.1.1.2 Warping Window Constraint

With classic DTW, pathological warping can occur when a small part of one subsequence is inappropriately mapped to a large portion of another [206]. To prevent this issue, one can constrain the amount of warping allowed. A popular approach is the Sakoe-Chiba Band (SCB)[218], which limits the maximum distance r the warping path can stray from the diagonal, where $\mid i - j \mid \leq r$.

[2]Each sequence is z-score normalized independently.

An additional benefit immediately apparent is that constraining the warping path can significantly speedup DTW because fewer elements are evaluated. The optimal warping window varies per problem [206], but a common constraint is 10% of the time series length, which also yields very good results in our testing.

6.1.1.3 Lower Bounding DTW

The cost of performing DTW is not much of a concern when working with segmented data at a low resampling rate as well as with a small number of gesture classes and training samples. However, when working with a continuous data stream where DTW evaluations are frequent and observational latencies are problematic, it can be useful to prune templates that will obviously not match a query. An efficient lower bound (LB) score, where $LB(T,Q) \leq DTW(T,Q)$, can be used to avoid a full evaluation in two cases: when the determined lower bound is worse than a predetermined rejection threshold or when a closer match has already been found. One such lower bound for DTW using ED is LB_{Keogh} [122], which envelopes a time series T with an upper (U) and lower (L) band based on a window constraint r:

$$\begin{aligned} U &= \left(\boldsymbol{u}_i = \max_{i-r \leq i \leq i+r} \boldsymbol{t}_i \;\middle|\; i = 1 \ldots T_N \right) \\ L &= \left(\boldsymbol{l}_i = \min_{i-r \leq i \leq i+r} \boldsymbol{t}_i \;\middle|\; i = 1 \ldots T_N \right), \end{aligned} \tag{6.13}$$

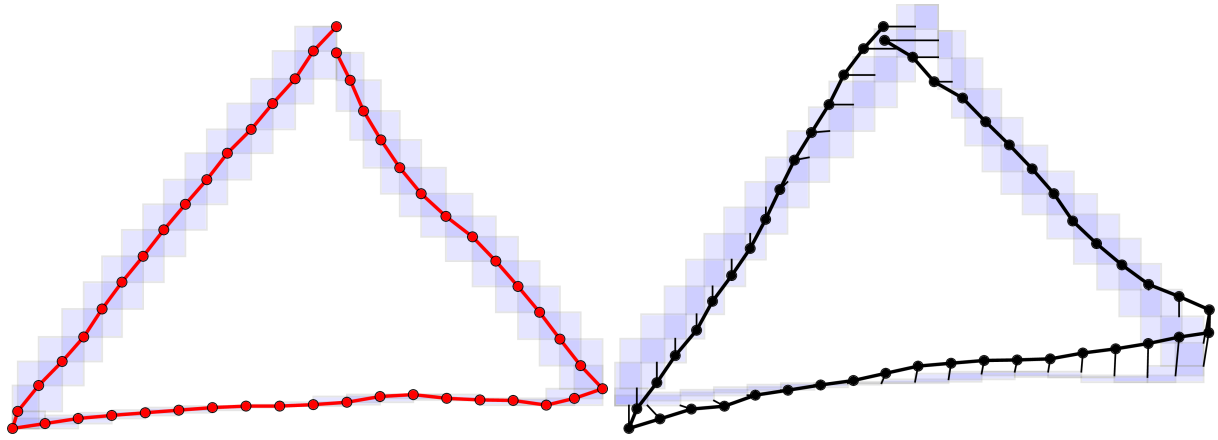


Figure 6.3: Visualization of a LB_{Keogh} lower bound in 2D for the triangle gesture from \$1-GDS [267]. On the left, the combined upper and lower bands of a template form a light blue bounding box per point. On the right, the LB_{Keogh} lower bound is calculated for a query as the sum of the minimum squared Euclidean distance from each point in Q to the corresponding point boundary from T (shown as thin black lines).

where max and min are component-wise operators, and $\forall_i \, \boldsymbol{l}_i \leq \boldsymbol{t}_i \leq \boldsymbol{u}_i$. To lower bound a query Q against a template T for DTW using ED, we define (again, component-wise):

$$LB_{Keogh}(Q,T) = \sum_{i=1}^{n} \begin{cases} (\boldsymbol{q}_i - \boldsymbol{u}_i)^2 & \text{if } \boldsymbol{q}_i > \boldsymbol{u}_i \\ (\boldsymbol{q}_i - \boldsymbol{l}_i)^2 & \text{if } \boldsymbol{q}_i < \boldsymbol{l}_i \\ 0 & \text{otherwise} \end{cases}, \tag{6.14}$$

which is visualized in Figure 6.3.

A lower bound has also been derived for the inner product of posteriorgrams [277] based on the local cost function $d(t_i, q_j) = -log \left\langle \boldsymbol{t}_i, \boldsymbol{q}_j \right\rangle$. One issue with this inner product lower bound is that each component in a posteriorgram is non-negative, which is unlike gesture path direction vectors whose components can take on negative values, and hence a different lower bound is required. Our inner product lower bound is similar to LB_{Keogh}, where the upper and lower bands are calculated

in the same way. However, the summation instead is based on the IP distance. If a component in the query vector is positive, then its similarity will be maximized when paired with the upper band component (if both signs are positive), but the dissimilarity is also minimized when paired with the upper band (if the signs are opposite). For the same reasons, if a component in a query vector is negative, pairing with the lower band will always produce the best possible result, which leads to the following IP lower bound:

$$LB_{IP}(Q,T) = \sum_{i=1}^{n} 1 - \min\left[1, \max\left(-1, < \hat{\boldsymbol{q}}_i, \boldsymbol{lb}_i >\right)\right]$$
$$\boldsymbol{lb}_i^j = \begin{cases} \boldsymbol{u}_i^j & \text{if } \boldsymbol{q}_i^j \geq 0 \\ \boldsymbol{l}_i^j & \text{otherwise} \end{cases}, \tag{6.15}$$

where j indexes a vector's components ($1 \leq j \leq m$) and the template and query sequence are assumed to be unit length. This lower bound can be proved using the same techniques as in [122, 277].

6.1.2 Correction Factors

After the DTW score is calculated, correction factors are applied that inflate this score based on additional dissimilarities not as easily perceived by only time warping measures, a method that we adopted from CIDDTW [15]. Our correction factors, however, use the inverse inner product of normalized feature vectors:

$$f_i(T,Q) = \frac{1}{< h_i(T),\ h_i(Q) >}, \tag{6.16}$$

where $2 \leq i \leq F$ per Equation 6.4 and each h_i transforms the time series into a *normalized* vector whose dimensionality is greater than one (otherwise its normalization would simply result in a scalar equal to one). The vector must also comprise only non-zero components so that the denom-

inator of Equation 6.16 falls in $(0,1]$ and the domain of f_i becomes $[1,\infty)$. Intuitively, times series that are similar should score near one so that DTW score inflation is minimized.

We now describe two correction factor transformations that are designed specifically for gestures and are inspired indirectly by Rubine's features for pen based gesture recognition [215]. In both cases, we focus on between-component information that is scale and position invariant. These latter attributes are important to ensure that differences between users do not appear as dissimilarities in their gesticulations. First, the component-wise absolute distance traversed by gesture X is given by:

$$\boldsymbol{h}^j_{abs} = \sum_{i=1}^{n-1} \left| \vec{\boldsymbol{x}}^j_i \right|, \tag{6.17}$$

Once $\boldsymbol{h}_{abs}$ is normalized, the components of this vector yield a relative measure of the total distance traveled by each component — this correction factor helps to ensure that contributions to the gesture path for each component of two different samples are similar in measure.

Our second feature is based on the bounding box extents of the concatenated *unnormalized* direction vectors in $\mathbb{R}^m$ space:

$$\boldsymbol{h}_{bb} = \boldsymbol{bb}_{max} - \boldsymbol{bb}_{min}, \tag{6.18}$$

where $\boldsymbol{bb}_{min}$ is the component-wise minimum of all points in a given trajectory and $\boldsymbol{bb}_{max}$ is the complementary maximum. Once $\boldsymbol{h}_{bb}$ is normalized, we have the relative distances spanned by each component. We found this bounding box correction factor to be useful in compensating for gestures that are mostly similar, except in span. Consider two gestures: *climb ladder*, which is a hand over hand action, and *balance*, which is a seesaw-like action with arms extended [63]. In both cases, the hands, wrists, and elbows oscillate vertically. The only major difference is that one's arms are extended outward to perform the balance gesture. This extension, which occurs at the beginning of the action, represents only a small duration of the entire gesture and may be difficult to detect with either ED or IP, depending exactly on how a participant performs either

gesture. However, the bounding box (span) of each gesture is very different, which is why using the bounding box as a correction factor can improve recognition accuracy.

6.2 Evaluation

In order to evaluate Jackknife, we consider a number of publicly available gesture datasets that span a range of input modalities. Where appropriate, we compare our recognizer against alternative methods. In general, we start with low dimensional data and work our way into higher dimensions. We perform repeated measures full factorial ANOVA analysis to understand if there are statistically significant differences between the recognizer variants evaluated. However, because recognition error rates are often quite low, accuracy measures are often non-normally distributed and may also violate the homogeneity of variance assumption, which is why we utilize the Aligned Rank Transform method for ANOVA analysis [265]. In the ANOVA results tables, the measure factor is either euclidean distance (ED) or inner product (IP), and the correction factors (CF) are either disabled (False) or enabled (True). Further, $T = x$ specifies that x samples (templates) per gesture class are used for training.

6.2.1 Pen and Touch

Table 6.1 presents results for Jackknife variants on the $1-GDS pen and touch dataset [267]. This dataset contains 4800 samples of 16 pen gestures collected from 10 participants. Our results were obtained using a writer-independent protocol. The reason for focusing on writer-independence is that state-of-the-art recognizers already achieve near perfect recognition rates in writer-dependent tests on this dataset. Given a training participant, our writer-independent protocol randomly selects T samples per gesture class from that participant for training. Next, one sample per each gesture

Table 6.1: Writer-independent mean accuracies for Jackknife variants on $1-GDS [267], as well as ANOVA results with statistically significant effects highlighted and their effect size labeled: (L)arge, (M)edium, or (S)mall.

Measure	**CF**	T=1		T=2	
		μ	(σ)	μ	(σ)
ED	False	0.91	(0.03)	0.92	(0.03)
ED	True	0.91	(0.03)	0.92	(0.03)
IP	False	0.94	(0.02)	0.95	(0.03)
IP	True	0.94	(0.02)	0.95	(0.03)

Effect	$F_{1,63}$	$Pr(>F)$	η_p^2
Measure	412.65	$< .001$	.87 (L)
CF	0.33	.57	.01 —
Templates	86.8	$< .001$	.58 (L)
Measure $\times$ CF	0.0	.96	.00 —
Measure $\times$ Templates	10.84	$< .01$	.15 (L)
CF $\times$ Templates	0.13	.72	.00 —
Measure $\times$ CF $\times$ Templates	0.02	.88	.00 —

Table 6.2: Writer-independent mean accuracies for several recognizers on $1-GDS [267]. There is a significant difference between recognizers with a large effect size ($F_{2,45}, p < .001, \eta_p^2 = .92$), and a post hoc analysis shows that all recognizers are significantly different from each other.

Recognizer	T=1		T=2	
	μ	(σ)	μ	(σ)
Jackknife	0.94	(0.02)	0.95	(0.03)
$1	0.88	(0.02)	0.90	(0.01)
$P	0.84	(0.02)	0.86	(0.02)

class is randomly selected from the pool of samples from all other participants, which is used as the test set. This sampling process is repeated 500 times, and all results for the training participant are combined into a single average accuracy value for that individual. This process is repeated for each participant, so that each is used to supply training data.

We observe that using inner products (IP) with gesture path direction vectors yields higher accuracy with significance, as does increasing the template count. Correction factors were not a significant factor; however we cannot say how the correction factors affect negative samples with this test. In Table 6.2 we compare the best Jackknife variant to $1 [267] and $P [250], two popular rapid prototyping $-family recognizers. The main observation is that Jackknife outperforms the alternative recognizers by a large margin and with significance.

6.2.2 Wii Remote

In Table 6.3, we report results for Jackknife variants on Cheema *et al.*'s Wii Remote dataset [38]. This dataset contains 15625 samples of gestures collected from 25 participants who individually provided 25 samples per gesture. This dataset is particularly interesting because as compared to other Wii Remote datasets, the gesture vocabulary is large at 25 classes and the Wii Remote traverses through various orientations. Our results were obtained using a user-dependent protocol similar to the original work of Cheema *et al.* [38]. For each experiment in this protocol, T samples are selected at random from a participant for training and the remaining $25 - T$ samples are selected for testing, and this selection process is performed 500 times per each participant. We did not run a user-independent protocol because device orientation has a significant impact on the accelerometer signal data, and there is a great deal of variance in how the Wii Remote is held by each participant. However, applications that track device orientation can easily adjust the signals in order to support user-independence. To improve accuracy for the inner product (IP) Jackknife variant, the acceleration data was integrated into 3D position points per [129], but we found this was unhelpful for the squared Euclidean distance variant (ED). There was a significant difference between the ED and IP measures, where the IP measure gave higher accuracies. There was also a small positive effect when utilizing the correction factors; though, due to truncation, this cannot be seen in the table.

Table 6.3: User-dependent mean accuracies for Jackknife variants on Cheema *et al.*'s Wii Remote dataset [38], as well as ANOVA results with statistically significant effects highlighted and their effect size labeled: (L)arge, (M)edium, or (S)mall.

Measure	**CF**	T=1		T=2	
		μ	(σ)	μ	(σ)
ED	False	0.79	(0.05)	0.86	(0.04)
ED	True	0.80	(0.05)	0.86	(0.04)
IP	False	0.93	(0.03)	0.96	(0.02)
IP	True	0.93	(0.03)	0.96	(0.02)

Effect	$F_{1,168}$	$Pr(>F)$	η_p^2
Measure	1747.23	$< .001$	.91 (L)
CF	3.91	$< .05$	.02 (S)
Templates	314.67	$< .001$	.65 (L)
Measure $\times$ CF	2.23	.14	.01 (S)
Measure $\times$ Templates	44.05	$< .001$	.21 (L)
CF $\times$ Templates	0.17	.68	.00 —
Measure $\times$ CF $\times$ Templates	0.03	.86	.00 —

Table 6.4: User-dependent mean accuracies for various recognizers on Cheema *et al.*'s Wii Remote dataset [38].The recognizer main effect is significant with a large effect size ($F_{3,168} = 789.72, p < .001, \eta_p^2 = .93(L)$), and post hoc tests show that all recognizers are also significantly different from each other.

Recognizer	T=1		T=2	
	μ	(σ)	μ	(σ)
DTW (w/ quantization)	0.94	(0.03)	0.97	(0.02)
Jackknife	0.93	(0.03)	0.96	(0.02)
$3	0.71	(0.06)	0.79	(0.06)
Protractor 3D	0.63	(0.06)	0.73	(0.06)

We next compared the best performing Jackknife variant against alternative domain-specific recognizers (DTW with quantization [146], $3 [128], and Protractor 3D [129]), whose results are shown in Table 6.4. DTW with quantized accelerometer data performed slightly better than Jackknife, although accuracies were similar. We note that other than the common step to integrate acceleration samples into position trajectories, Jackknife did not require domain specific knowledge to achieve high accuracy. On the other hand, DTW with quantized data required an analysis of the accelerometer data and a selection of the quantization levels, which may be domain or device specific.

6.2.3 Kinect

In Table 6.5, results are presented for the Ellis *et al.* (Parkour) [63] Kinect dataset, which contains 1280 samples of 16 parkour actions, *e.g.* climbing and vaulting, collected from 16 participants using a Kinect sensor. We used the same user-dependent test protocol as reported in the last section. The local cost function was significant, where the Jackknife IP measure outperformed ED, and IP was able to achieve 99% accuracy with one training sample. The correction factors were are also significant and played a role in substantially driving down the error rates[3]. We also ran a user-independent variant of this test and found that IP with correction factors and T=2 achieved 96% accuracy, whereas ED with correction factors achieved 70%.

We also replicated Ellis *et al.*'s observational latency test [63] that evaluated recognizer accuracy when training and test data were truncated to varying frame counts in order to minimize the delay between when a user performs an action and when the time that action is recognized. If an action can be recognized before completion, observational latency can be reduced. In Table 6.6, we see

[3]It is important to note that as accuracies reach high levels, seemingly small improvements in accuracy are actually large reductions in error rates.

Table 6.5: User-dependent mean accuracies for Jackknife variants on the Parkour dataset [63], as well as ANOVA results with statistically significant effects highlighted and their effect size labeled: (L)arge, (M)edium, or (S)mall.

Measure	**CF**	T=1		T=2	
		μ	(σ)	μ	(σ)
ED	False	0.88	(0.05)	0.93	(0.03)
ED	True	0.89	(0.05)	0.94	(0.03)
IP	False	0.97	(0.03)	0.99	(0.02)
IP	True	0.99	(0.02)	0.99	(0.01)

Effect	$F_{1,105}$	$Pr(>F)$	η_p^2
Measure	383.89	$< .001$	.79 (L)
CF	16.62	$< .001$	.14 (M)
Templates	80.36	$< .001$	.43 (L)
Measure $\times$ CF	1.73	.19	.02 (S)
Measure $\times$ Templates	35.86	$< .001$	.25 (L)
CF $\times$ Templates	3.38	.07	.03 (S)
Measure $\times$ CF $\times$ Templates	0.0	.96	.00 —

Table 6.6: Recognition percentage accuracies for Jackknife and recognizers evaluated by Ellis *et al.* [63] (bottom three) for varying length video sequences. Both training and testing data are truncated to the specified number of frames.

Recognizer	10	15	20	25	30	40	60
Jackknife IP w/ CF	24	53	78	90	95	98	99
Ellis *et al.* [63]	14	37	65	82	91	95	96
Conditional Random Field	15	25	47	67	81	91	94
Bag of Words	11	21	44	68	83	92	94

that Jackknife with IP achieves the highest accuracy for all frame count levels.

6.2.4 Acoustic Gestures

Hand based gesture interactions with computers via Doppler shifted sound waves is presently gaining attention [88, 213]. Unlike other interface devices evaluated to this point, sound waves are especially subject to noise as extracting and detecting frequency changes over a large spectrum with low cost hardware still lacks robustness. Further, complex over-the-air hand gestures are prone to large variations in shape and speed, the later of which manifests itself uniquely in frequency distributions (unlike in 2D or 3D Euclidean space where speed alone does not change the observed trajectory). These issues make accurate gesture recognition difficult. To test whether Jackknife is suitable for this input modality, we collected a new dataset comprising eighteen dynamic hand gestures collected from 22 participants with a 5 speaker, 1 microphone setup based on [190] (see our project website for more details). The raw data is represented as a 33 frequency bin distribution per speaker, which results in an $m = 165$ component vector per frame. Jackknife results are shown in Table 6.7, which were obtained using the user-dependent protocol [38] described previously. In early testing, we learned that z-score normalization on the spectrum data was harmful; we believe this is because for some gestures, there is no motion through certain frequency bins, and so z-score normalizing those components only served to scale up noise. Therefore, ED- is also reported, which is the squared Euclidean distance measure on raw data without z-score normalization. While ED- is not formally part of Jackknife (as this result required an additional investigation and some domain knowledge), we feel that the reader should be aware of this result. Additionally, since the bounding box correction factor does not have meaning in this context, we only utilize the absolute distances traveled correction factor.

As is shown, the local distance measure and correction factor effects are significant. In a post hoc analysis, we found that ED- was significantly different from ED and IP, but the latter measures where not different from each other. Good accuracy ($\geq 90\%$) can be achieved with ED- with two

Table 6.7: User-dependent accuracy results for Jackknife variants on the acoustic gesture dataset, as well as ANOVA results with statistically significant effects highlighted and their effect size labeled: (L)arge, (M)edium, or (S)mall. ED- indicates that sequences were not z-score normalized.

Measure	**CF**	T=1		T=2		T=4	
		μ	(σ)	μ	(σ)	μ	(σ)
ED	False	0.71	(0.08)	0.81	(0.08)	0.87	(0.07)
ED	True	0.75	(0.07)	0.83	(0.06)	0.89	(0.05)
ED-	False	0.83	(0.07)	0.90	(0.05)	0.94	(0.04)
ED-	True	0.84	(0.06)	0.91	(0.05)	0.94	(0.04)
IP	False	0.72	(0.07)	0.81	(0.06)	0.88	(0.06)
IP	True	0.75	(0.06)	0.84	(0.05)	0.90	(0.05)

Effect	F Value	$Pr(>F)$	η_p^2
Measure	$F_{2,483} = 491.46$	$< .001$	.67 (L)
CF	$F_{1,483} = 72.36$	$< .001$	.13 (M)
Templates	$F_{3,483} = 551.31$	$< .001$	.77 (L)
Measure $\times$ CF	$F_{2,483} = 10.02$	$< .001$	.04 (S)
Measure $\times$ Templates	$F_{6,483} = 11.41$	$< .001$	.12 (M)
CF $\times$ Templates	$F_{3,483} = 1.76$	.15	.01 (S)

templates, otherwise four templates are required with IP.

6.2.5 Performance

Over 10,000 iterations, we trained a C++ based Jackknife recognizer with our Kinect user study data (see next section), which is 63 components per point. The recognizer was trained with T=10 templates per gesture class, resulting in 140 templates (since there are 14 gesture classes). The recognizer was then used to evaluate one additional sample per gesture class and the evaluation time in *micro*seconds per sample was recorded. On a MacBook Pro, 2.2 GHz Intel Core i7 with 8GB DDR3 memory, the average time to execute a recognition test on a raw sample was 395 µs (std=90.2); or equivalently, the evaluation process took approximately 2.82 µs (0.64) per template.

6.3 Discussion

Jackknife is designed to be a "go-to" recognizer for gesture interface development that can span multiple modalities and still achieve competitive accuracy. By combining DTW with concepts developed for 2D gesture recognition, for rapid prototyping and gesture customization, we believe our recognizer accomplishes its objective. There is no simple criterion by which one can judge a recognizer and determine that is it excellent or terrible. At best we can compare with other recognizers in a specific domain for a specific dataset when such is available and speak in relative terms. In this way, we see that Jackknife is very competitive. Examples include the \$1-GDS pen and touch dataset where Jackknife with IP outperformed \$1 and \$P; Cheema *et al.*'s [38] Wii Remote datasets, where Jackknife was on par with the domain-specific quantized DTW; and Ellis *et al.*'s [63] Parkour dataset, where our recognizer achieved 99% accuracy with one training sample and can be used to reduce observational latency. Even with atypical gesture data such as sound frequencies, Jackknife is able to reach greater than 90% accuracy with only two training samples per gesture. From another perspective, based on accuracy results reported for 36 3D gesture recognizers across different domains [137] (Table 1), one might expect a competitive recognizer to fall between 88—98% ($93\% \pm 5.29\%$) accuracy. In all tests performed, Jackknife fell in this range.

6.3.1 One Recognizer To Rule Them All?

No, while Jackknife has been demonstrated to be a versatile tool, we do not claim that our recognizer is appropriate for every situation or even that it is the best solution possible for any one situation. For this reason, some limitations are worth noting. First, Jackknife presently does not handle static poses. Motion into and out of a static pose can be detected, but a single frame pose does not constitute a time series appropriate for DTW treatment, a limitation of Jackknife we intend to address in future work. The techniques presented also do not generalize to arbitrary variability;

for instance, a 3D hand gesture of a circle drawn counterclockwise does not automatically pair with a clockwise gesture without providing appropriate training samples.

6.4 Conclusion

We have presented Jackknife, a general gesture recognizer suitable for rapid prototyping and gesture customization that works well with little training data. Our recognizer uses DTW with an inner product local cost function on normalized gesture path direction vectors, which was shown to outperform the squared Euclidean distance alternative in most tests, although Jackknife can be equipped with either measure. We also introduced two new correction factors for DTW that help disambiguate certain gesture classes and inflate negative, non-gesture sample scores. Overall, we have demonstrated that Jackknife shows promise of being a capable and robust go-to gesture recognizer.

CHAPTER 7: CUSTOM GESTURE SEGMENTATION WITH MACHETE[1]

President Joe once had a dream
The world held his hand, gave their pledge

So he told them his scheme for a Savior Machine
They called it the Prayer, its answer was law

David Bowie
Saviour Machine

The Dollar General (TDG) uses a hierarchical approach to continuous gesture recognition, where one segments incoming data into gesture candidates and a high quality recognizer classifies or rejects each candidate gesture. In this chapter we present a new device-agnostic segmentation technique called Machete, which adheres as much as possible to $-family principles. Machete builds off of Penny Pincher and Jackknife by exploiting the direction vector representation of human motion in an elastic matching framework. Although a gesture recognition pipeline can use Machete with any underlying recognizer, we use Jackknife.

[1]This chapter contains material to appear in the following article: Taranta II, Eugene M., *et al.* "Machete: Easy, Efficient, and Precise Continuous Custom Gesture Segmentation." ACM Transactions on Computer-Human Interaction (TOCHI) 2020. See Appendix C for associated copyright information.

7.1 Machete

7.1.1 Preprocessing

Machete works on direction vectors, which means we must first transform input signal X into a series of deltas as follows:

$$\vec{X} = (\vec{\boldsymbol{x}}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i \mid i = 1 \ldots X_N - 1)\,. \tag{7.1}$$

During this transformation, we discard new input samples that would produce invalid vectors. For instance, we discard samples that are less than one pixel away from the last accepted position when working with mouse data. It is also critical that one smooths the input signal with a low-pass filter so as to remove jitter and variabilities in the direction vector signal. Small differences in position over time can result in significant fluctuations between consecutive direction vectors. And although we account for this issue via a per input sample weighting scheme, filtering nonetheless further improves performance.

7.1.1.1 Training

Given demonstration sample X belonging to gesture class $g_i \in \mathbb{G}$, we must convert the sample into a Machete template. To do this, we first find a suitable low resolution representation of X using a modified DP line simplification algorithm, after which we then define template T as the normalized direction vectors between DP points Y:

$$T = \left(\vec{\boldsymbol{t}}_{\boldsymbol{i}} = \frac{\boldsymbol{y}_{i+1} - \boldsymbol{y}_i}{||\boldsymbol{y}_{i+1} - \boldsymbol{y}_i||} \;\middle|\; i = 1 \ldots Y_N - 1 \right). \tag{7.2}$$

7.1.2 Dynamic Time Warping

Note, this discuss replicates Chapter 6.1.1 but is included again here for the reader's convenience. Dynamic time warping (DTW) is an elastic matching dissimilarity measure that finds the optimal alignment between two sequences based on a given local cost function. Although squared Euclidean distance is the most common cost function, Jackknife [235] utilizes the inner product of gesture path direction vectors, which provides translation and scale invariance. We adopt this approach herein not just because of its simplicity, but because we are unaware of any alternative computationally efficient data representation or technique that can handle position and scale variance on a per time-step basis.

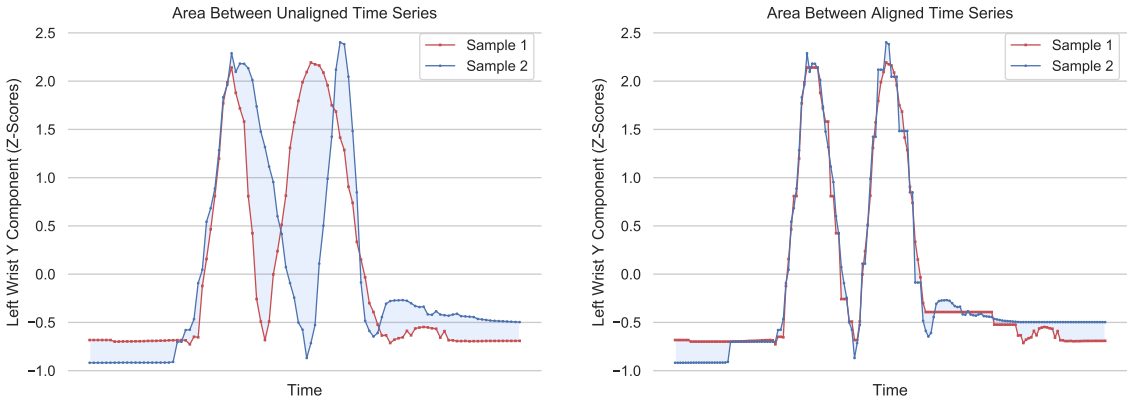


Figure 7.1: Visualization of two time series before and after DTW alignment. The shaded regions between them represent the amount of dissimilarity measured by Euclidean distance. Although the unaligned series are of a shorter duration, the area between them after alignment is significantly improved.

To understand how DTW works at a high level, consider Figure 7.1 above. We present two sequences produced by the same individual who twice in a row raises their arm up high. In the left graph, notice that the sequences are temporally out of sync because of differences in production speed. The shaded area between each sequence measures the squared Euclidean distance, *i.e.*, $\sum (x_i - y_i)^2$ between them, which reveals major dissimilarities. Contrast this with DTW which is

able to resolve temporal differences as shown in the right graph, where we see that DTW eliminates most of the shaded area. How does this work? Internally, DTW constructs an optimal *warping path* W that specifies how each element from query Q maps to template T. Warping path W is defined as:

$$W = \Big(\boldsymbol{w}_i = (a,b) \;\Big|\; 1 \le a \le |Q|, \quad 1 \le b \le |T| \Big), \tag{7.3}$$

and is used by DTW in the following way:

$$DTW(Q,T) = \sum_{i=1}^{|W|} d(\boldsymbol{q}_a, \boldsymbol{t}_b) \tag{7.4}$$

$$a = \boldsymbol{w}_{i_a}$$

$$b = \boldsymbol{w}_{i_b}$$

where $d(\boldsymbol{q}_a, \boldsymbol{t}_b)$ is a local cost function that measures the dissimilarity between query element $\boldsymbol{q}_a$ and template element $\boldsymbol{t}_b$.

Next we need to understand how DTW constructs the warping path. Let us suppose we have partial warping path $W[1:N]$ and the last element therein $\boldsymbol{w}_N = (i,\ j)$ maps query $\boldsymbol{q}_i$ to template $\boldsymbol{t}_j$. From here, we have three options for how we can proceed—we can set:

- $\boldsymbol{w}_{N+1} = (i,\ j+1)$ Repeat query $\boldsymbol{q}_i$ so that it is matched with template $\boldsymbol{t}_{j+1}$
- $\boldsymbol{w}_{N+1} = (i+1,\ j)$ Repeat template $\boldsymbol{t}_j$ so that it is matched with query $\boldsymbol{q}_{i+1}$
- $\boldsymbol{w}_{N+1} = (i+1,\ j+1)$ Repeat neither and match query $\boldsymbol{q}_{i+1}$ with template $\boldsymbol{t}_{j+1}$

It is by repeating elements in either sequence that DTW is able to align the sequences. Referring back to Figure 7.1 (right), one can see example short plateaus in the aligned curves where elements are repeated.

One might note that there are many possible warping paths that align two sequences, but DTW selects the optimal solution via an elegant dynamic programming approach. DTW constructs a $|Q| \times |T|$ matrix M, where each element (i, j) holds the optimal warping path score that aligns $Q[1 : i]$ with $T[1 : j]$. In other words, once two subsequences are aligned, element (i, j) is the measure between them. The matching decision described above can then be decided using the following recurrence relation:

$$M_{i,j} = d(\boldsymbol{q}_i, \boldsymbol{t}_j) + min \begin{cases} M_{i,j-1} & \text{Repeat query } \boldsymbol{q}_i \\ M_{i-1,j} & \text{Repeat template } \boldsymbol{t}_j \\ M_{i-1,j-1} & \text{Repeat neither,} \end{cases} \tag{7.5}$$

with boundary conditions:

$$M_{0,0} = 0 \tag{7.6}$$

$$M_{i,0} = \infty \tag{7.7}$$

$$M_{0,j} = \infty. \tag{7.8}$$

In code, one can fill in the matrix one row at a time from left to right, starting at $(1, 1)$. Then for each subsequent element (i, j), one evaluates Equation 7.5. Once this process reaches the bottom right corner $(|Q|, |T|)$, the optimal path (and score) through the matrix is known. Two important DTW properties worth noting are that every warping path through M forces $\boldsymbol{q}_1$ to be matched with $\boldsymbol{t}_1$ (as a consequence of the boundary conditions), and similarly $\boldsymbol{q}_{|Q|}$ is matched with $\boldsymbol{t}_{|T|}$.

Pathological warping occurs when a small section of one sequence is aligned with a large section of a second sequence due to unbound warping [207]. To prevent pathological warping, one may impose a warping path constraint, such as a Sakoe-Chiba band [218]. And one may also use lower

bounding [122, 235, 277] along with other techniques [201] to optimize the alignment process, but these optimizations do not apply to this work.

7.1.3 Continuous Dynamic Programming

Continuous dynamic programming (CDP) [179] extends DTW so as to be able to match patterns embedded in unsegmented sequences, including gestures embedded in continuous signals. Specifically, CDP replaces the initial matching constraint imposed on DTW by Equation 7.8 with a new condition $M_{i,0} = 0$. In this way, CDP introduces the start of a new warping path every frame that extends on through matrix M only when its cumulative distance is sufficiently low. We present an example of this process in Figure 7.2. When the system samples a new Kinect pose $\boldsymbol{q}_i$ at time i, CDP establishes the start of a new path by matching pose $\boldsymbol{q}_i$ with template $\boldsymbol{t}_1$. CDP then processes the remainder of the row as standard DTW would. At the end of the update, $M_{i,|T|}$ holds the optimal warping path score ending at time i, and by monitoring this score we can speculate about whether the user gesticulated. We illustrate this idea in the previous figure where we highlight the optimal warping path through the matrix associated with the best score found in the last column, and we further visualize the eight matched elements. After having found a sufficiently low score, our system can extract the subsequence and pass it to the underlying recognizer for further analysis.

Since DTW has been used in prior work that meets our design criteria [146, 235], and CDP is a straightforward extension of DTW as just described, we decided to use continuous dynamic programming in Machete. However, standard CDP suffers from scale and position invariance under the Euclidean distance measure, which we address in our work. Specifically, in the following subsections we (1) develop a new local cost function, (2) define CDP_{ip}, and (3) introduce a new initial matching constraint.

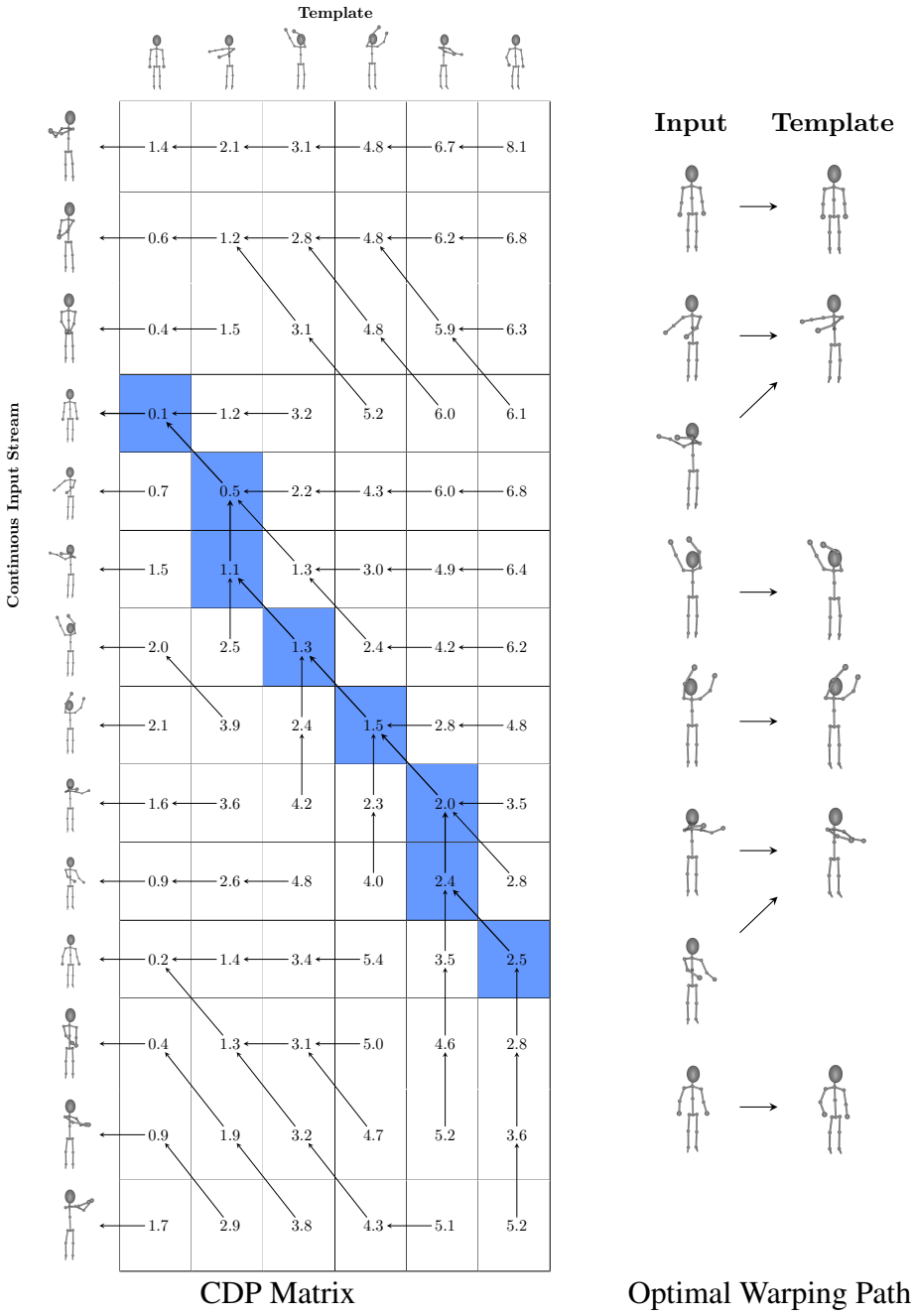


Figure 7.2: Visualization of a continuous dynamic programming matrix. Each row is the result of processing one new input sample from a continuous stream, and each element holds the accumulated warping path score. Arrows indicate the direction of each warping path through the matrix. The blue warping path highlights the best result, showing which input poses where matched with which template poses.

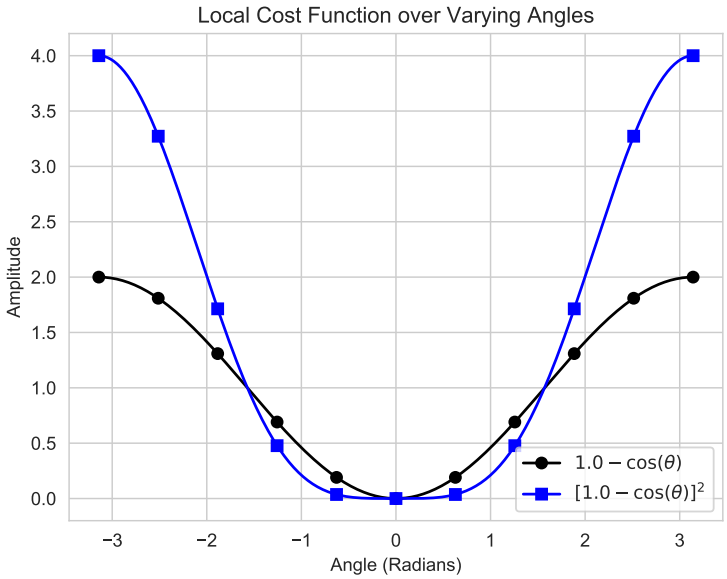


Figure 7.3: Amplitude of unweighted local cost functions over varying angles. The squared variant ensures that a wider variability of similar angles are mapped to lower costs, whereas dissimilar angles are more severely penalized.

7.1.3.1 Local Cost Function

The squared Euclidean distance (ED) local cost function one typically employs in DTW and CDP is inappropriate for several continuous gesture recognition problems. Namely, squared ED is neither position nor scale invariant. One way around this issue is to use a running z-score normalization scheme [201], but finding an appropriate lag in the presence of large discontinuities (such as when a person jumps to a new location on an interactive display) may prove challenging. Further, we want to avoid scaling noise to unit variance which can happen when idle components do not change position during gesticulation, *e.g.*, the y-component in a minus sign. A simpler approach is to measure the inner product between direction vectors, which is invariant to position and scale and has already been used successfully to recognize gestures across varying input device types [235,

238]. We specifically leverage the inner product between direction vectors as follows:

$$d\left(\vec{\boldsymbol{q}}_i, \vec{\boldsymbol{t}}_j\right) = \left(1 - \frac{\left\langle \vec{\boldsymbol{q}}_i, \vec{\boldsymbol{t}}_j \right\rangle}{\|\vec{\boldsymbol{q}}_i\|}\right)^2. \tag{7.9}$$

In words, the inner product of normalized direction vectors fall in the interval $[-1,1]$. Since template elements are already normalized, we only need to divide $\left\langle \vec{\boldsymbol{q}}_i, \vec{\boldsymbol{t}}_j \right\rangle$ by the query's vector length $\|\vec{\boldsymbol{q}}_i\|$. To transform this product into a dissimilarly measure, we subtract it from one, and as a consequence, the score falls in the interval $[0,2]$. We further square the sum to allow for a greater variability in low angular differences, and to further penalize severely diverging trajectories. This difference can be seen in Figure 7.3. Certainly one can use other powers, but squaring the sum is computationally efficient as it does not involve an expensive power function call, and we found that cubing the sum did not improve performance.

7.1.3.2 Continuous Dynamic Programming with Direction Vectors

Given Equation 7.9 as the local cost function, we define CDP_{ip} as the *weighted* sum of the optimal warping path through M, which corresponds to the recurrence blow. Note, our unique contributions are, first, how we weight the local cost function with $\|\vec{\boldsymbol{q}}_i\|$ and, second, how we track and normalize the result with path length $L_{i,j}$:

$$M_{i,j} = \frac{1}{L_{i,j}} \left[\|\vec{\boldsymbol{q}}_i\| \cdot d\left(\vec{\boldsymbol{q}}_i, \vec{\boldsymbol{t}}_j\right) + L_{k,l} \cdot min \begin{cases} M_{i,j-1} \\ M_{i-1,j} \\ M_{i-1,j-1} \end{cases} \right] \tag{7.10}$$

$$L_{i,j} = L_{k,l} + \|\vec{\boldsymbol{q}}_i\|. \tag{7.11}$$

L is an auxiliary matrix that stores the sum of query vector lengths along each warping path. And indices (k,l) correspond to the selected matrix element $M_{k,l}$ (being one of $M_{i,j-1}$, $M_{i-1,j}$ or $M_{i-1,j-1}$). The effect of Equation 7.10 on the overall cost is that the contribution of each warping path element is weighted by the query vector's length. This treatment ensures that short vectors occurring on cusps and corners have less impact on the final score than do vectors that define the main body of motion. It is worth noting that we informally tested CDP_{ip} without the normalization factor and saw a significant drop in performance.

We designed our warping path formula with $-family principles in mind. The local cost function is weighted by the input direction vector length, a simple geometric operation that has been used in prior work [232, 235, 238]; and we track the full gesture path length through the warping path, which is merely an accumulator. Similarly, in code, it is trivial to define a warping path object that stores the start frame, unnormalized weighted cumulative score, and cumulative path length. In this way, one can calculate $M_{i,j}$ on the fly by normalizing the score and extend a warping path by propagating these values to the next element, as shown in our pseudocode in Appendix B.

7.1.3.3 Boundary Conditions

One issue with matching direction vectors is that once a gesture begins, a subsequent vector may better match the first template element. When this happens, the new warping path will overtake the prior path rather than extend it. Our solution to overcome this problem is to modify the initial matching condition so that CDP_{ip} only starts a new warping path when the present path passing through the first column is worse than a threshold θ. Formerly, CDP_{ip}'s boundaries conditions are:

$$\begin{aligned} M_{0,0} &= 0 & L_{0,0} &= 0 \\ M_{i,0} &= (1 - cos(\theta))^2 & L_{i,0} &= 0 \\ M_{0,j} &= \infty & L_{0,j} &= 1 \end{aligned}$$

Note that because $L_{i,0} = 0$, the transition from column zero to one, which starts a new warping path, carries zero cost. With this modification, vectors matching the first template element are now grouped together. In Section 7.2 we discuss how we select the θ parameter in further detail.

7.1.4 Correction Factors

One limitation of Machete is that we are unable to localize gesture end points that occur within a straight line. For full body gestures, this issue is less problematic, but straight lines are common for 2D gestures. Our sink node weighting scheme enables Machete to latch onto the start point when the gesture boundary occurs on a curve, but we also need a similar mechanism for end points. To

address end point localization, we adopt the concept of correction factors [15, 235]:

$$CDP'_{ip} = CDP_{ip} \times cf_{openness} \times cf_{f2l}. \tag{7.12}$$

The adjusted CDP'_{ip} score is the standard score inflated by two correction factors that measure dissimilarity information not captured by CDP_{ip}. Specifically, we look at the relationship between the first and last points of a gesture, $cf_{openness}$. When they are far apart, the gesture is open, and conversely, the gesture is closed when these points are collocated. Second, we also consider the angle made by the direction vector between them, cf_{f2l}. Since the direction vector may be unreliable when a gesture is closed but provide useful information about correct gesture production when open, we use a weighting scheme to balance these two factors, which we describe shortly. A correction factor resolves to one (zero inflation) when there is no difference between the query and template, and differences between them cause the factors to scale upward, though we limit each factor's maximum value to two (not shown in the math below).

We limit the design of our new correction factors to the same set of operations used in prior rapid prototyping work [232, 235, 238] to ensure compliance with our design goals; namely we use scalars, vectors, ratios, weighting, min-max functions, and inner products. In greater detail, let us define the first to last point (abbreviated $f2l$) direction vector as follows:

$$\vec{\boldsymbol{x}}_{f2l} = \boldsymbol{x}_l - \boldsymbol{x}_f, \tag{7.13}$$

where f and l respectively index the gesture's start and end points. From $\vec{\boldsymbol{x}}_{f2l}$, we derive the openness of a gesture as the length of its direction vector over the gesture's path length:

$$openness = \frac{\|\vec{\boldsymbol{x}}_{f2l}\|}{\mathscr{L}}. \tag{7.14}$$

Here, $\mathscr{L}$ is the total path length $X[f:l]$, which we measure directly from training data or estimate with $L_{l,|T|}$ for queries. We use these values because of their fast computation—one can index the points from a circular buffer and $L_{l,|T|}$ is readily available. Intuitively, from Equation 7.14, we see a five point star gesture will score low in openness, whereas a gesture separated by the length of its boundary will score highest. With training sample's $\vec{\boldsymbol{t}}_{f2l}$ direction vector, we also decide how much to weight the $cf_{openness}$ and cf_{f2l} factors. The direction vector length relative to the gesture's bounding box diagonal length *diag* gives us a sense of the gesture's openness, where a value close to one means the gesture is fully open. Note, we cannot use the diagonal length in the above definition of openness because we found the per component boundaries were too expensive to carry forward through time. However, to calculate the bounding box during training is okay. With this in mind, we calculate the weights with:

$$w_{f2l} = \min\left(1,\ 2\ \frac{\|\vec{\boldsymbol{t}}_{f2l}\|}{diag}\right) \tag{7.15}$$

$$w_{closedness} = 1 - \frac{\|\vec{\boldsymbol{t}}_{f2l}\|}{diag}. \tag{7.16}$$

By these definitions, we give preference to w_{f2l} when the gesture is more open and $w_{closedness}$ when the gesture is closed.

With the direction vectors, openness, and weights, we last define the correction factors as follows:

$$cf_{f2l} = 1 + w_{f2l}\ \frac{1}{2}\left(1.0 - \left\langle \frac{\vec{\boldsymbol{q}}_{f2l}}{\|\vec{\boldsymbol{q}}_{f2l}\|}, \frac{\vec{\boldsymbol{t}}_{f2l}}{\|\vec{\boldsymbol{t}}_{f2l}\|} \right\rangle\right) \tag{7.17}$$

$$cf_{openness} = 1 + w_{closedness}\left(\frac{\max\left(openness(Q), openness(T)\right)}{\min\left(openness(Q), openness(T)\right)} - 1.0\right). \tag{7.18}$$

Without weighting, the cf_{f2l} factor is manipulated so that it evaluates to one when the query and template direction vectors match, whereas cf_{f2l} evaluates to two when they perfectly differ. The

fraction in $cf_{openess}$ is a ratio greater than one that describes the difference in openness. Similarly, $cf_{openness}$ is one when their openness matches, but scales to higher values as their openness diverges.

These correction factors together help compensate for situations where gestures end on a long line. In our evaluation, we only apply these factors to mouse data, where we have several such gestures. For full body interactions, however, we do not leverage this mechanism, and it is a goal of our future work to derive a more general solution.

7.1.5 Pruning

Machete generates a new gesture candidate every frame, but in order to reduce the computational burden of calling an underlying recognizer too frequently, we wish to prune as many candidates as possible. Figure 7.4 illustrates how several CDP_{ip} template scores progress over time during a cartwheel right gesture. We observe that most (though not all) curves stay relatively high while the user swings their arms, whereas the cartwheel template score drops close to zero. We anticipate this drop for two reasons. First, we expect well defined gestures to stand out from other human motion, which implies that the input will diverge from most template patterns at least enough to drive the score high on average. Second, because we square the inner product, sufficiently similar patterns ought to approach zero. Based on these assumptions, we are able to derive a relatively efficient pruning strategy.

For each template, we track the running average and automatically reject any candidate that is above half the mean. We believe this is a reasonable conservative estimate given that scores are expected to approach zero during gesticulation. To further improve pruning, we also assume that gesture end points correlate with minima. Based on this idea, we also prune any candidate that does not occur on a minimum. This rule results in the automatic rejection of most remaining candidates.

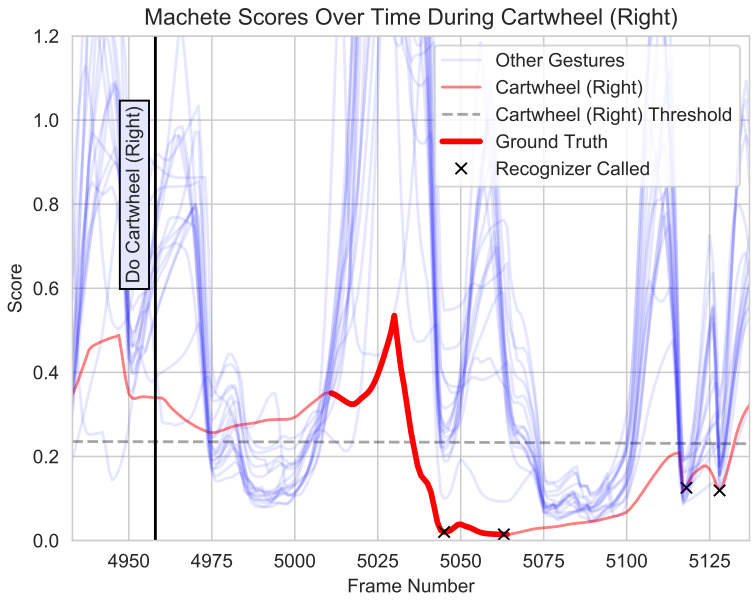


Figure 7.4: Example Machete segmentation scores over time as a participant performs the cartwheel right gesture. Seventeen templates are loaded. Per frame Machete scores are rendered red for the cartwheel gesture and blue for the remaining sixteen gestures. Ground truth segmentation is in bold red. Based on our pruning strategy, the four minimum values below the template threshold (horizontal dashed black line) are where candidate gestures are passed from Machete to an underlying recognizer for further evaluation. Notice that in approxiamtely 200 frames, the recognizer is invoked only four times to analyze the cartwheel gesture.

As shown in Figure 7.4, only a few candidate gestures satisfy these requirements, whereas the vast majority are discarded.

7.1.6 Angular DP

Our final point of order is to describe how Machete transforms a given training sample into a template. Many recognizers uniformly resample the sample's trajectory to a fixed number of points that are equidistantly spaced along its arc length. However, uniform resampling may cut corners, express the gesture's shape with too many points, or both. Instead, we hope to reduce computation

by representing a pattern with the least number of points possible, while preserving important features. One technique that does both is the Douglas-Peucker (DP) line simplification algorithm [55], which has been successfully used in gesture recognition [78].

The algorithm proceeds as follows. Given line segment $\overline{\boldsymbol{x}_a\boldsymbol{x}_b}$, we calculate the distance of each point $\boldsymbol{x}_i,\ a < i < b$ to the segment. We then select the furtherest point $\boldsymbol{x}_i$ from the segment and subdivide it into two halves: $\overline{\boldsymbol{x}_a\boldsymbol{x}_i}$ and $\overline{\boldsymbol{x}_i\boldsymbol{x}_b}$. DP recurses into both halves and continues on as such until either the furthest distance falls below a predetermined threshold ε or the segment is indivisible. All selected points are then combined to describe a simplified version of the original trajectory. This approach is generally effective, but because Machete works on direction vectors, we make a small modification. Instead of using distance directly, we weight it by the normalized

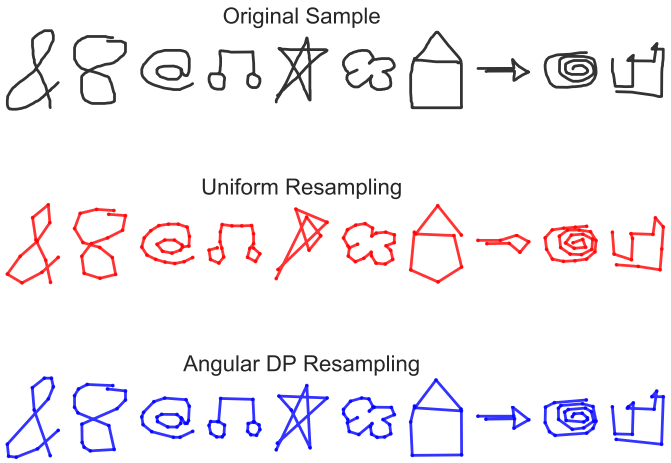


Figure 7.5: Comparison of resampling techniques. Each original sample is resampled using angular DP and then uniformly resampled using the same number of points.

angle it forms with the end points:

$$d_i' = d_i \cdot \arccos\left(\frac{\langle \vec{v}_1, \vec{v}_2 \rangle}{\|\vec{v}_1\| \cdot \|\vec{v}_2\|}\right)\frac{1}{\pi} \tag{7.19}$$

$$\vec{v}_1 = \boldsymbol{x}_i - \boldsymbol{x}_a \tag{7.20}$$

$$\vec{v}_2 = \boldsymbol{x}_b - \boldsymbol{x}_i, \tag{7.21}$$

where d_i is the distance from $\boldsymbol{x}_i$ to $\overline{\boldsymbol{x}_a\boldsymbol{x}_b}$ and d_i' is the scaled distance. Please see our pseudocode in Appendix B for more information.

We chose to weight the distance of a point by its associated angle in order to help DP select those points where the trajectory changes direction, *e.g.*, on corners and cusps. We believed this would lead to better simplified gesture representations using fewer points, which we later confirm. For now, we illustrate the impact of Angular DP on resampling in Figure 7.5. For each sample, we first resample the trajectory using Angular DP with $\varepsilon = .01$. We then uniformly resample the original trajectory to the same number of points selected by Angular DP. By inspecting their shapes side by side, one will notice that angular DP provides a better description of each gesture shape, often using relatively few points.

7.2 Parameter Selection

Our system requires three parameters, a low-pass filter cutoff frequency f_c, an angular DP termination threshold ε (Section 7.1.6), and an initial sink node cost θ (Section 7.1.3.3). We turn to the former first, but before we can select a cutoff frequency, we must choose a low-pass filter. There are several options compatible with \$-family principles, including the moving average, exponential moving average, double exponential moving average [136], and 1€ [35] filters. We decided to use

a multiple-pass centered moving average (MCMA) because of its simplicity, optimal properties [225], and because we wanted the filtered response to correspond directly to the original input to facilitate analysis with our development tools. The CMA filter has a single parameter M, the number of points per average, which we set to three:

$$\boldsymbol{y}_i = \frac{\boldsymbol{x}_{i-1} + \boldsymbol{x}_i + \boldsymbol{x}_{i+1}}{3}, \tag{7.22}$$

where signal Y is the filtered response over input signal X. MCMA builds on this by passing the signal through a CMA filter R times. Algebraic manipulation of basic MCMA facts using our window size leads to:

$$R = \frac{3}{2}\left(\frac{f_s}{2\pi f_c}\right)^2. \tag{7.23}$$

The sampling rate f_s can be determined from training data or at runtime, but the cutoff frequency f_c is less trivial. In general, most human motion falls below 10 Hz [276], though this varies over different types of motion. Although there are automatic low-pass filter calibration procedures for pointing tasks [230] which balance precision and lag, we are unaware of similar procedures for gestures. So to help us decide appropriate cutoff values, we examined two public datasets that are representative of the gesture types used in our evaluation. The first is a Kinect dataset [63] comprising sixteen full body parkour gestures collected from sixteen participants, totaling 1280 samples. The second, \$1-GDS [267], is a pen and touch gesture dataset comprising sixteen gestures articulated at three different speeds by ten participants, totaling 4800 samples.

We conducted power spectral density (PSD) analysis on each sample and combined their cumulative powers into a single distribution. These results are shown in Figure 7.6. Our analysis reveals that information embedded in full body gestures fall below 3 Hz, which we select as the cutoff frequency f_c for our Kinect and Vive input described later. For pen and touch gestures, we see that

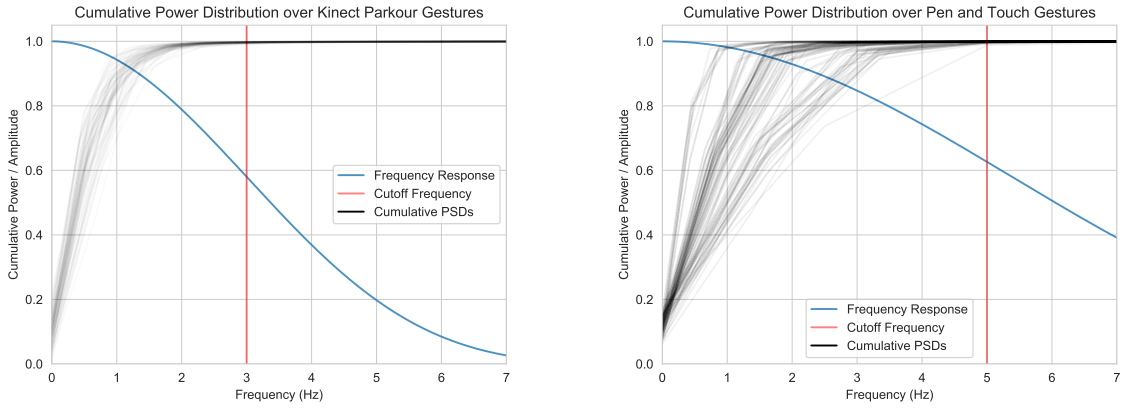


Figure 7.6: Cumulative power spectral distribution for parkour Kinect (left) as well as pen and touch (right) gesture signals. Vertical red lines denotes our proposed cutoff frequency for full body and hand gestures, respectively.

most information falls below 5 Hz, which we select as the cutoff frequency f_c for mouse input. In addition to PSD data, we also illustrate MCMA's frequency response for each cutoff, which shows the reduction in signal strength over varying frequencies. Specifically, this shows what ratio of each frequency will remain in the signal after being filtered. The strength of low frequencies are left mostly unaltered, whereas high frequencies are almost entirely removed. Recall that our main objective with filtering is to remove high frequency noise that jitters the direction vectors without distorting the gesture's trajectory. In this regard, one will note that MCMA has a slow rolloff; so even if the cutoff is not optimal, there is still sufficient wiggle room and the gesture shape will largely remain in tact.

Next we must determine an appropriate threshold ε that balances precision and computation. Our goal is then to find ε that will adequately describe complex trajectories. To do this we compare the intraclass and interclass score distributions of a sufficiently complex dataset. Specifically, for a given dataset and threshold ε, we compute the pairwise dissimilarity of each sample to every other sample in the dataset using Machete—between two samples, one is used as a template and

the other is treated as continuous input. For each class g_1 and g_2, we find g_1's intraclass mean score and the interclass mean score from g_1 to g_2, and then we save their ratio. A ratio greater than one means the classes are perfectly separated, whereas a ratio less than one indicates there may be confusion between the classes under Machete, and to have the best separation possible, we select threshold ε that maximizes the ratio between the closest classes.

We carried out our analysis over two pen datasets combined into one whole [257]. The Execution Difficulty Sets (EDS) were created to model and test the perceived difficulty of articulating unistroke pen gestures. In aggregate there were thirty-eight gestures articulated twenty times, with eighteen gestures being produced by fourteen participants and the remaining twenty by eleven, for a grand total of 9440 samples. We chose to use EDS because compared to other options, they varied in complexity. With a smaller dataset, we run the risk of selecting a poor threshold because classes are easily separated and not representative of all motions possible in its associated input space.

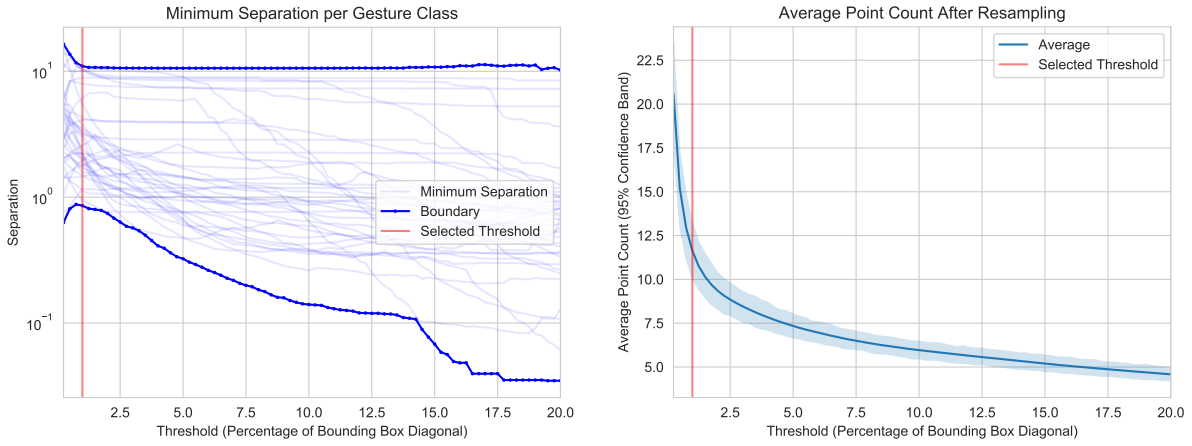


Figure 7.7: *Left*: Minimum separation between each EDS [257] gesture class and its nearest neighbor as measured by Machete over varying ε thresholds. *Right*: Average resampling rate N per threshold.

Using the described procedure on EDS, we collected the minimum separation ratio between classes

over varying thresholds. In Figure 7.7, we plot the result for each of the thirty-six classes as well as the average resampling rate N for each threshold. We find that the best separation occurs at $\varepsilon = 0.75\%$, which we round up to 1% since the difference is not substantial. At this level, we also note that the resampling rate is low compared to typical values used by $-family recognizers, *e.g.* $Q recommends using $N = 32$. Although lower rates are possible, we see that the separation between classes also quickly drops, and so we are compelled to remain at $\varepsilon = 1\%$

With a ε threshold in hand, we were able to next decide on the initial sink node cost θ. For this parameter, we simply resampled all of the samples of a given dataset using angular DP, and thereafter measured the pairwise angles between each sample within the same class. We again used EDS for hand gestures, and analysis revealed the average angle to be $\mu = 20.8$ ($\sigma = 32.4$), which we rounded to 20 degrees. Informal testing on our mouse dataset revealed this was a good choice as θ values outside this locality produces worse results. For fully body gestures, using the same procedure over the Kinect parkour dataset, we found that θ jumped to nearly $\mu = 65.02$ ($\sigma = 27.0$) degrees. Informal testing revealing that although results were reasonable, this threshold was more liberal than necessary. When we lowered the parameter to $\theta = 40$ degrees, performance slightly improved and so we used this value throughout our evaluation for full body gestures.

7.3 Data Collection

To evaluate the effectiveness of Machete against alternative techniques, we collected high-activity data from thirty participants over three input device types (ten participants per device).

7.3.1 Data Collection

We designed Machete to spot custom gestures among other high-activity interactions. However, because most public datasets were captured with a single device and comprise presegmented sequences, low-activity interactions, or both, we decided to collect four new high-activity datasets using three unique input device types via a game-based data collection protocol. Our goal was to ensure that participants engaged in a range of activities that vary beyond just gestures, such as to puppeteer an avatar or directly manipulate objects in a virtual environment. To this end, we designed a simple "Follow The Leader" (FTL) game that forces players to remain active between gesticulations in a way that is consistent with our objective. Through the interface, we present a continuously active *leader* who performs various random movements that participants must mimic to their best ability. At random times throughout the game, we further present text commands that require a player to break solidarity and gesticulate, see Figure 7.8. This strategy results in a rich stream of complex, high-activity motions that (unlike a typical one-at-a-time data collection protocol) captures participants easing into and out of gesticulations from other activities. Another reason we chose this approach is because gesture productions may differ when a participant's attention is divided. For instance, several studies show a significant difference in recognizer accuracy when comparing game data to homogenous cross validated data [40, 236, 238], which we believe is also likely to make segmentation an even more difficult problem.

7.3.1.1 Apparatuses

We chose to leverage the Kinect, Vive, and mouse for a number of reasons. Namely, they cover a wide range of input device types, sensor qualities, and data representations. Concerning Kinect, we recorded twenty-one 3D skeletal joint positions using an XBOX ONE Kinect, for a total of $m = 63$ components per data point. However, since the hand and foot data were especially noisy,

we zeroed out this part of the signal. The HTC Vive reports two 3D controller positions and orientations via quaternions, which we break into two separate datasets, for a total of $m = 6$ and $m = 8$ components per data point, respectively. Last, a mouse reports 2D position data, thereby yielding $m = 2$ components per data point (since we do not sample the button state signals). We also chose to evaluate mouse data because it is challenging to perform complex trajectories with this input device due to issues with friction and range, whereby participants must constantly reposition their arm, hand, and device.

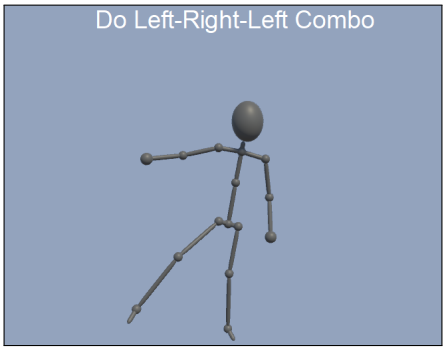


Figure 7.8: An example Follow the Leader (FTL) screenshot: The leader and participant are standing on one foot, as if to maintain balance on a tightrope, when a gesture command pops up. At this time, the participant will immediately stop following and execute the command, after which they will return to following the leader.

7.3.1.2 Follow The Leader

We developed FTL in Unity[2] as a means to procure Kinect, HTC Vive controller, and mouse data. FTL has two modes, which are the demonstration and game modes. In demonstration mode, we record five custom samples of each gesture class in a randomized order so as to increase within class variability[3]. For Kinect and Vive, we first display text that specifies what gesture the partici-

[2]Unity is a popular, real-time game development platform, see https://unity.com/.

[3]Gestures repeated back-to-back may have less variation.

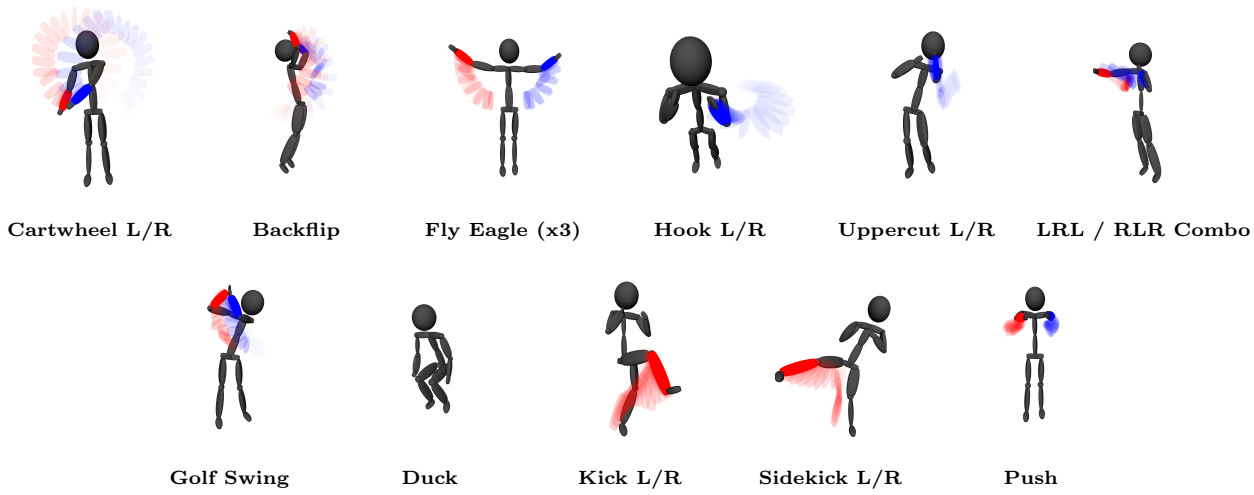


Figure 7.9: High activity gesture dataset for Kinect and Vive, where L and R denote left and right respectively. For example, there is a left upper cut gesture and a right uppercut gesture. All seventeen gestures were used in our Kinect evaluation, where only the first eleven (top row and golf swing) were used for Vive.

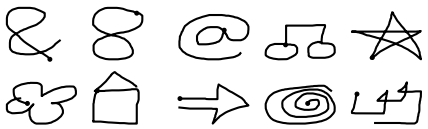


Figure 7.10: The ten mouse gestures used in our evaluation, taken from [257, 267].

pant must perform, and once he or she indicates their readiness, we start a countdown timer. When this timer reachers zero, we commence recording and conclude the record via a keyboard stroke after the gesture is complete. For mouse data, a participant instead toggles the left mouse button to achieve the same effect. We replay each sample to verify correctness and ensure the participant is satisfied with their result. When a participant performs a gesture incorrectly or a tracking error occurs, we simply discard the faulty sample and try again. In total, we collected three datasets comprising 17 full-body Kinect, 11 HTC Vive controller, and 10 mouse gestures. See Figures 7.9 and 7.10 for a depiction of each.

In FTL's game mode, we render a leader[4] on screen and participants are told to follow its motion as closely as possible. All avatar movements are non-gesture actions designed to mimic the direct manipulation and puppeteering kind of interactions that may occur in a video game, however, at unknown intervals we display a text command instructing the participant to perform a specific gesture. During gesticulation, we visually confirmed that the participant correctly produced the gesture, as there were some instances when a participant forgot a gesture's form or confused left with right. If we observed an error, the instance was re-queued to be retried at a later time. We collected three correct instances of every gesture throughout each game session. Before we began the main session, however, we first allowed each participant to complete a practice round, where each gesture was executed one time, thereby allowing a participant to become familiar with FTL's mechanics. Each practice session lasted approximately two minutes.

7.3.1.3 Subjects

We recruited thirty students (17 male, 13 female) from a local university to participate in our study. Their ages ranged from 18 to 29 with a median age of 22. All but one participant owned a game system, and none of the participants had any mobility problems. For each participant, we first explained the study's purpose and then demonstrated each gesture they would have to perform, after which we started to collect data. Each session lasted about 30 minutes and participants were allowed to take a break if they felt tired.

[4]With Kinect and Vive, the leader is a skeleton model, whereas with mouse, the leader is a long mouse trail.

7.4 Evaluation of Angular DP

To understand whether angular DP is a viable alternative to standard DP and uniform resampling, we examined the effect of resampling rate on all training data under an inner product measure. For a given training sample and rate N, we spatially resampled the training sample to N points using DP, angular DP, and uniform resampling. Then to measure differences between a training sample and its resampled variants, we again uniformly resampled all four trajectories to a high resolution ($N = 1024$), and measured the inner product between corresponding normalized direction vectors:

$$f\left(\vec{X},\vec{Y}\right) = \sum_{i=1}^{1024} \langle \vec{\boldsymbol{x}}_{\boldsymbol{i}}, \vec{\boldsymbol{y}}_{\boldsymbol{i}} \rangle \,. \tag{7.24}$$

Dissimilarity measure f informs us of the resampled trajectory's likeness to its original form. We next relate the three methods to each other by measuring the percentage improvement of angular DP and standard DP over uniform resampling. This improvement is equivalent to the percentage decrease in dissimilarity:

$$\text{Improvement} = \left(1 - \frac{\text{DP}}{\text{Uniform}}\right) \times 100\%, \tag{7.25}$$

where DP is the dissimilarity measure for a given DP method, and Uniform is the same measure under uniform resampling.

We finally averaged together all results per dataset per N as shown in Figure 7.11. Across all conditions, angular DP showed an average improvement of 44% ($\sigma = 24\%$) over uniform resampling, whereas standard DP only achieved an average improvement of 37% ($\sigma = 24\%$). Further, on visual inspection, we see the direction of this difference is consistent for all values of N, across

all datasets, except for especially low resampling rates where all three methods are approximately equivalent. In our view, these results justify our minor modification to DP when working with direction vectors.

7.4.1 Automatic Parameter Selection

As part of our evaluation, we also recorded the resampling rate N selected by our automatic parameter selection algorithm for angular DP (Section 7.1.6) for each training sample. These results are shown in Fig 7.11, which are grouped by gesture class and sorted by median. Across all conditions, the average resampling rate was $N = 13$ ($\sigma = 4.6$), whereas the minimum and maximum values were respectively 4 and 30. Compared with common rates found in the literature (*e.g.*, the recent \$Q uses $N = 32$), these results are quite low, which is expected to boost Machete's computational performance. Further, results are consistent with our intuition that longer and more complex gestures require higher N to adequately describe its shape.

7.5 Evaluation of Machete

We compare Machete against three alternative segmentation approaches to determine which technique is most performant in both accuracy and computation across all four datasets. We assume that a given segmenter is paired with a robust recognizer having a reliable accept-reject criteria. In this way, the segmenter may frequently pass candidate gestures to the recognizer without concern for generating false positives, and the recognizer in turn will evaluate each candidate, accepting only those queries that are most like their gesture class while rejecting any query that does not match. In practice, the recognizer may incorrectly accept or reject certain candidates or notify an application of gesticulation before it is complete, even as a better solution simultaneously unfolds.

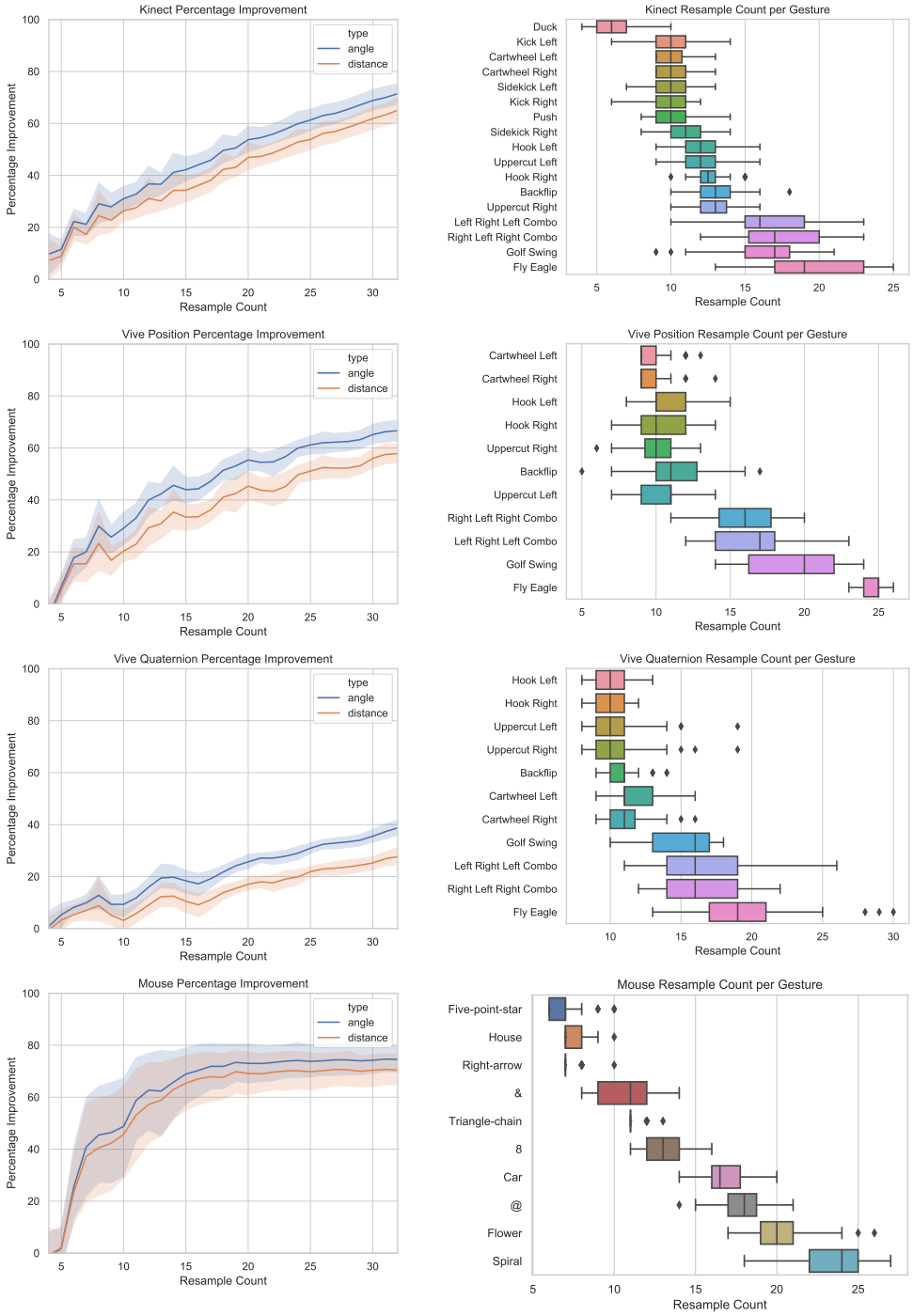


Figure 7.11: Left: Percentage improvement in self similarity measure relative to uniform resampling for standard DP using distance only and our angular DP variant (higher is better). Right: Resample count distribution per gesture, sorted by median.

However, for the purpose of our first evaluation, we focus mainly on how well our segmenter is able to localize a gesture's end points.

Specifically, we pair each segmenter with Jackknife [235], a general purpose, device-agnostic custom gesture recognizer. For a given participant and training sample, we train both the segmenter and Jackknife with the specified sample, after which we replay the session one frame at a time through the system. After each call, the segmenter returns a flag indicating whether or not it has identified a gesture candidate. If so, the candidate is fed to the recognizer as a query where it is scored. When playback is within the boundaries of a section where FTL told a participant to perform a gesture, we log the best score and segmentation result.

7.5.1 The Contenders

As described in our Related Work chapter, we identified three segmentation approaches that we believe are compliant with $-family principles: an energy-based method, windowing, and CDP. The former identifies gesture boundaries by examining a time series's energy profile. Given input time series X, we define its energy profile as:

$$E = (e_i \mid i = 1 \ldots |X| - 1) \tag{7.26}$$

$$e_i = \frac{1}{2}||\boldsymbol{x}_{i+1} - \boldsymbol{x}_i||^2. \tag{7.27}$$

We treat each minimum in E as a candidate gesture boundary. As such, when we observe a new boundary in a continuous stream, we assume it correlates with the end of a gesture, and its beginning can be any previous boundary that falls within 1.5 times the training sample's length. We then pass each start-end pair as a candidate gesture to the underlying recognizer for further analysis. It may be possible for us to reduce the number of candidate gestures by rejecting boundaries whose

energies are too high, but this requires learning a threshold from training data, which is difficult to do in a customization context.

Windowing is perhaps the most popular method and was included because of its ubiquity and effectiveness. In our evaluation, we assume that the training sample length provides the best approximation of duration in practice. Therefore, we match our window length to the training sample. One can instead use multiple windows to improve segmentation, but this will increase computational costs, which is contrary to our objective.

Finally, since Machete is based on continuous dynamic programming, we also wanted to evaluate standard CDP. One issue, however, is that CDP measures Euclidean distance, which is neither position nor scale invariant and consequently inappropriate for our evaluation. For this reason, we replace its local cost function with an inner product measure that works on normalized direction vectors, though the result is not squared as it is with Machete. Further, we normalize the accumulated cost by warping path length, not distance, as we do for Machete. Note, without these two changes, we found CDP was unusable with our HA datasets.

7.5.2 Preprocessing and Ground Truth

Since a training sample is likely to contain idle data at its tails, we trim each Kinect and Vive sample as part of a preprocessing step. This step was especially helpful in controlling the sliding window length, which will have otherwise been too long. Specifically, to remove unwanted frames, we computed the cumulative energy over time and removed all frames that fall outside of the inner 98%—one percent of energy was cut from each tail. What remains is a valid pattern that a given segmenter must match. We chose to use this process given that it is sufficiently straightforward and

anyone can use it without difficulty in practice. Second, because mouse data may contain hooks[5], after angular DP processing, we heuristically cut the first and last vector when their length was less than 20% of their neighboring vectors. Further, because there is a great deal of variability in mouse gesture production, we additionally train the system with samples rotated ± 15 degrees—thus, each training sample is converted into three templates for all segmenters.

For each gesture instance in a session, we must establish ground truth segmentation and identify where in time the gesture begins and ends. To accomplish this task, we use a two step process. First, we conduct a naive brute force search in the region of a session where an instance is known to reside. Specifically, if we expect to find a gesture instance between times t_1 and t_2, we search for the start s and end e times that minimize the dissimilarly between subsequence $X_{s,e}$ and a given training sample Y under measure f:

$$s, e = \underset{t_1 \leq s \leq e \leq t_2}{\operatorname{argmin}} \; f\left(X_{s,e}, Y\right), \tag{7.28}$$

where f is a DTW-based measure that varies per device type and yields a reasonable segment approximation. Since there are five training samples per gesture class, we further take the median of the aggregate start and end results as an estimate of ground truth. Finally, in step two, we examine each result from step one. If we observe an obvious error, we manually search for the correct solution and log the updated result. However, we try to limit such corrections as this second step is highly subjective.

[5]Changes in direction due to imprecision that sometimes occur at the start or end of a stroke.

7.5.3 Error Measures

Given our evaluation protocol, we obtained one segmentation result per instance per training sample that in aggregate form 7350 results—10 participants $\times$ 5 training samples per participant $\times$ 3 instances per session $\times$ (17 Kinect + 11 Vive position + 11 Vive quaternion + 10 mouse) gesture classes. We examined each result to determine if the given segmenter correctly detected the associated gesture instance. For each miss, we removed the offending result along with the remaining three corresponding segmentation results, so that the population remained identical between methods. With those results that remained, we proceeded to extract four error measures. The first three of these were the signed start, signed end, and total segmentation errors. Let G_s, G_e respectively denote the ground truth start and end times, and let S_s, S_e similarly denote the segmenter candidate boundaries. The three errors are then defined as:

$$\text{Start Error} = S_s - G_s \tag{7.29}$$

$$\text{End Error} = S_e - G_e \tag{7.30}$$

$$\text{Total Error} = |\text{Start Error}| + |\text{End Error}|\,. \tag{7.31}$$

These measures are important in understanding how well a method temporally segments gestures, but alone the result can be misleading. When one gesticulates quickly, the difference between two frames can have a significant impact on the gesture's shape, whereas this difference is less dramatic at lower speeds. For this reason, we also measure the arc length error:

$$\text{Arc Length Error} = \frac{\mathscr{L}\left(X\left[\min(S_s, G_s) : \max(S_s, G_s)\right]\right) + \mathscr{L}\left(X\left[\min(S_e, G_e) : \max(S_e, G_e)\right]\right)}{\mathscr{L}\left(X[G_s : G_e]\right)}, \tag{7.32}$$

where X is the session time series, and $\mathscr{L}$ is the arc length:

$$\mathscr{L}(X) = \sum_{i=2}^{|X|} \|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\| . \tag{7.33}$$

As one can see, we relate clipped and extended boundaries to the gesture's total arc length via a ratio. If the start and end errors for a given gesture instance are respectfully one and zero frames, but this difference represents 5% of total gesture's arc length, then the arc length error is 5%, which we believe better represents how well a segmenter captured the gesture's shape, compared to time-based error measures. To gain an intuition of this measure, we visualize varying arc length error magnitudes in Figure 7.12. Roughly, with errors that are 10% or less, we see that the gesture shape is well preserved. Beyond this point, clipping and extensions begin to fundamentally change gesture shapes, which can lead to an increase in false negatives when the underlying recognizer is unable to identify an otherwise well articulated gesture.

7.5.4 Design and Analysis

We conducted one experiment per dataset, where each experiment was a 1-factor repeated measures design. The nominal factor was *segmenter*, for which there were four levels: Machete, CDP, Window, and Energy. The outcome variables were those just discussed—the start, end, total, and arc length errors. These errors were averaged over each participant into a single result. We then used Friedman testing [77] to drive our omnibus, after which we conducted post-hoc analysis with exact, two-tailed Wilcoxon signed rank tests, while protecting against multiple comparison type I errors by way of the Holm-Bonferroni step-down procedure [100].

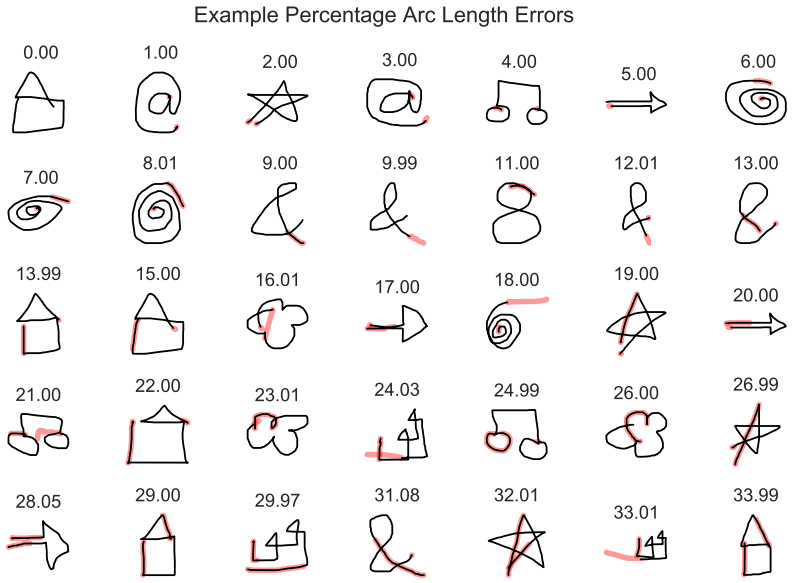


Figure 7.12: Varying arc length errors over mouse data. We render ground truth as a thin black line and highlight errors in red where the segmenter truncated a gesture or extended beyond its temporal boundary.

7.5.5 Results

7.5.5.1 Temporal Variability

We first examined the temporal variability of ground truth results as shown in Figure 7.13. These results represent the pairwise interclass distribution of duration, defined as the ratio of maximum to minimum duration between each pair of gesture instances belonging to the same class and participant. We included this data specifically to give one a sense of how well we might expect the window method to do over the varying datasets. Vive gestures have the least variation ($\mu = 1.08, \sigma = 0.08$), which is followed by Kinect ($\mu = 1.11, \sigma = 0.11$) and mouse ($\mu = 1.27, \sigma = 0.28$) gestures. Indeed we see by arc length error analysis (discussed soon), that the windowing segmenter's performance declines across the datasets in accordance with this variability. In most

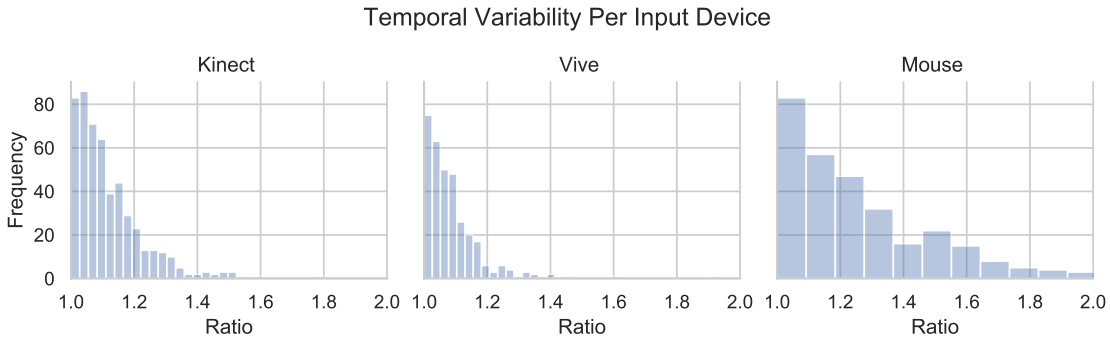


Figure 7.13: Distribution of pairwise intraclass ground truth duration ratios.

cases, performance is still acceptable, but it could be improved by using a multiple window scheme.

7.5.5.2 Computational Performance

In addition to segmentation errors, we also recorded the average processing time per frame needed to run each segmenter, which includes any calls to the underlying Jackknife recognizer. Average nanosecond times are presented in Table 7.1. We see that Machete and CDP are extremely fast compared to window and energy-based segmentation. The latter is especially slow because it calls Jackknife numerous times when it identifies a local minimum in the energy profile. Since we believe that windowing is the present first-choice method for segmentation, we use this technique as a baseline to calculate the percentage decrease in processing time as presented in Table 7.2. Machete reduces the computational burden by at least 98% across all experiments. CDP performs similarly well, but because its accuracy is subpar (as will be shown), we do not consider CDP a practical option. In contrast, energy-based segmentation runs thousands of times worse than window based segmentation, which implies one is better off running multiple sliding windows in parallel.

In this work, we are concerned primarily with computational performance because as discussed in

Table 7.1: Average (μ) per frame processing time in nanoseconds and standard deviation (σ).

	Kinect		Vive Position		Vive Quaternion		Mouse	
Method	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)
Machete	1.71	(0.48)	0.54	(0.18)	0.57	(0.16)	2.63	(0.73)
CDP	6.55	(0.86)	1.61	(0.31)	1.81	(0.34)	6.28	(0.97)
Window	98.44	(17.19)	44.58	(11.92)	53.94	(11.39)	123.84	(19.91)
Energy	13445.53	(2035.94)	6507.65	(2661.44)	9646.43	(3952.72)	6836.84	(1777.55)

Table 7.2: Percentage decrease in computation time per segmenter invocation using Window as the baseline.

Method	**Kinect**	**Vive Position**	**Vive Quaternion**	**Mouse**
Machete	98.27	98.80	98.95	97.88
CDP	93.35	96.39	96.65	94.93
Window	0.00	0.00	0.00	0.00
Energy	-13558.13	-14498.28	-17782.85	-5420.48

Chapter 2, fast processing allows us to process more templates per unit time, enable segmentation on low resource devices, and free up system resources; and as shown, Machete is remarkably efficient. However, with only one template loaded, we cannot afford to sacrifice segmentation accuracy, and, as we see next, Machete works well under this condition.

7.5.5.3 Kinect Segmentation

Friedman tests revealed a significant difference between the four segmenters across all error measures. On average, Machete achieved the lowest error as shown in Table 7.3. Concerning start segmentation error, Machete was significantly different from CDP and energy, but not window, per Table 7.4. This was true for the remaining measures as well. Window was also different from energy, and energy from CDP across the end segmentation and total error measures. These results

Table 7.3: Average (μ) segmentation error in frames and standard deviation (σ) for Kinect position data, as well as the percentage path length error.

	Start Error		End Error		Total Error		Length Error	
Method	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)
Machete	2.41	(0.65)	3.45	(1.05)	5.86	(1.68)	7.45	(1.75)
CDP	4.50	(0.77)	3.70	(1.15)	8.21	(1.89)	12.48	(1.56)
Window	3.42	(0.84)	4.94	(1.26)	8.36	(1.74)	8.17	(1.92)
Energy	4.42	(1.21)	6.86	(1.80)	11.28	(2.48)	10.23	(2.18)

Table 7.4: Kinect segmentation post-hoc results using the exact, two-tailed Wilcoxon signed rank test. Friedman test results are also listed under each column header.

	Start Error $\left(\chi^2_3 = 20.0, p < .001\right)$			End Error $\left(\chi^2_3 = 22.2, p < .001\right)$			Total Error $\left(\chi^2_3 = 22.0, p < .001\right)$			Length Error $\left(\chi^2_3 = 22.6, p < .001\right)$		
Pair	$W+$	p	r	$W+$	p	r	$W+$	p	r	$W+$	p	r
Machete - Window	4	0.05	—	6	0.05	—	3	$< .05$	0.9 L	16	0.28	—
Machete - CDP	55	$< .05$	1.0 L	55	$< .05$	1.0 L	55	$< .05$	1.0 L	55	$< .05$	1.0 L
Machete - Energy	0	$< .05$	1.0 L	0	$< .05$	1.0 L	0	$< .05$	1.0 L	3	$< .05$	0.9 L
Window - CDP	49	0.08	—	9	0.06	—	26	0.92	—	55	$< .05$	1.0 L
Window - Energy	47	0.10	—	54	$< .05$	1.0 L	52	$< .05$	0.9 L	46	0.15	—
CDP - Energy	35	0.49	—	0	$< .05$	1.0 L	5	$< .05$	0.8 L	47	0.15	—

in sum suggest that Machete and window-based segmentation are both reasonable approaches, though Machete may be a better option because of its significantly lower total segmentation error.

7.5.5.4 Vive Position Segmentation

Results for Vive position data are very similar to Kinect. Friedman tests revealed a significant difference between the four segmenters across all error measures. On average, Machete again achieved the lowest error as shown in Table 7.5. On the start, total, and arc length error, Machete was significantly different from CDP and energy, but not window, per Table 7.6; but Machete was not different from CDP in end error. Like with Kinect, these results suggest that Machete

Table 7.5: Average (μ) segmentation error in frames and standard deviation (σ) for Vive position data, as well as the percentage path length error.

	Start Error		End Error		Total Error		Length Error	
Method	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)
Machete	8.95	(2.45)	10.65	(2.87)	19.60	(4.56)	6.10	(1.33)
CDP	16.61	(2.32)	11.68	(3.05)	28.29	(3.83)	13.93	(1.16)
Window	10.83	(3.50)	13.11	(2.90)	23.95	(5.84)	6.93	(0.89)
Energy	17.00	(3.60)	22.93	(5.43)	39.93	(7.64)	8.31	(1.94)

Table 7.6: Vive position segmentation post-hoc results using the exact, two-tailed Wilcoxon signed rank test. Friedman test results are also shown under each column header.

	Start Error $\left(\chi^2_3 = 22.4, p < .001\right)$			End Error $\left(\chi^2_3 = 24.2, p < .001\right)$			Total Error $\left(\chi^2_3 = 23.5, p < .001\right)$			Length Error $\left(\chi^2_3 = 22.4, p < .001\right)$		
Pair	$W+$	p	r	$W+$	p	r	$W+$	p	r	$W+$	p	r
Machete - Window	14	0.39	—	1	$< .05$	1.0 L	6	0.05	—	9	0.13	—
Machete - CDP	55	$< .05$	1.0 L	48	0.07	—	55	$< .05$	1.0 L	55	$< .05$	1.0 L
Machete - Energy	0	$< .05$	1.0 L	0	$< .05$	1.0 L	0	$< .05$	1.0 L	4	$< .05$	0.9 L
Window - CDP	53	$< .05$	0.9 L	12	0.13	—	49	0.05	—	55	$< .05$	1.0 L
Window - Energy	55	$< .05$	1.0 L	55	$< .05$	1.0 L	55	$< .05$	1.0 L	46	0.13	—
CDP - Energy	24	0.77	—	0	$< .05$	1.0 L	1	$< .05$	1.0 L	54	$< .05$	1.0 L

and window-based segmentation are both reasonable approaches, though Machete may be a better option because of its significantly lower end segmentation error. We also note that their difference in the end error measure was marginal.

7.5.5.5 Vive Quaternion Segmentation

Friedman tests revealed a significant difference between the four segmenters across all error measures. On average, Machete for the third time achieved the lowest error as shown in Table 7.7, although there were less significant differences (see Table 7.8). In start and total error, Machete differed from CDP and energy, and on arc length error, Machete differed from CDP. Across the

Table 7.7: Average (μ) segmentation error in frames and standard deviation (σ) for Vive quaternion data, as well as the percentage path length error.

	Start Error		End Error		Total Error		Length Error	
Method	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)
Machete	9.02	(1.83)	10.72	(2.10)	19.75	(2.75)	7.32	(0.76)
CDP	12.94	(2.70)	11.27	(2.14)	24.20	(3.31)	12.15	(1.47)
Window	12.71	(5.13)	14.09	(3.94)	26.80	(7.96)	7.85	(1.29)
Energy	17.82	(4.58)	17.17	(3.46)	34.99	(6.01)	8.51	(1.11)

Table 7.8: Vive quaternion segmentation post-hoc results using the exact, two-tailed Wilcoxon signed rank test. Friedman test results are also listed under each column header.

	Start Error $\left(\chi^2_3 = 19.1, p < .001\right)$			End Error $\left(\chi^2_3 = 17.8, p < .001\right)$			Total Error $\left(\chi^2_3 = 20.3, p < .001\right)$			Length Error $\left(\chi^2_3 = 21.0, p < .001\right)$		
Pair	$W+$	p	r	$W+$	p	r	$W+$	p	r	$W+$	p	r
Machete - Window	7	0.11	—	4	0.05	—	5	0.06	—	16	0.55	—
Machete - CDP	54	$< .05$	1.0 L	46	0.13	—	55	$< .05$	1.0 L	55	$< .05$	1.0 L
Machete - Energy	0	$< .05$	1.0 L	0	$< .05$	1.0 L	0	$< .05$	1.0 L	6	0.08	—
Window - CDP	33	0.62	—	7	0.11	—	26	0.92	—	55	$< .05$	1.0 L
Window - Energy	49	0.11	—	45	0.13	—	49	0.06	—	39	0.55	—
CDP - Energy	7	0.11	—	0	$< .05$	1.0 L	0	$< .05$	1.0 L	55	$< .05$	1.0 L

four measures, Machete did not differ from window-based segmentation, but we note that the end and total errors are marginal. Again, like with Kinect and Vive, these results suggest that Machete and window-based segmentation are both reasonable approaches, but unlike in prior cases, Machete does not have a clear lead over windowing in error.

7.5.5.6 Mouse Segmentation

Friedman tests revealed a significant difference between the four segmenters across all error measures. For the first time, energy achieved the lowest arc length error, which was significantly different from all other methods (see Tables 7.9 and 7.10). However, Machete still achieved the lowest

Table 7.9: Average (μ) segmentation error in frames and standard deviation (σ) for mouse position data, as well as the percentage path length error.

	Start Error		End Error		Total Error		Length Error	
Method	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)
Machete	19.86	(8.68)	24.90	(4.38)	44.76	(7.68)	5.61	(1.31)
CDP	35.35	(8.73)	25.27	(3.87)	60.62	(9.66)	13.98	(2.12)
Window	28.75	(7.72)	33.57	(4.88)	62.31	(9.14)	10.08	(2.46)
Energy	25.72	(7.77)	26.84	(3.51)	52.56	(8.46)	4.73	(1.36)

Table 7.10: Mouse segmentation post-hoc results using the exact, two-tailed Wilcoxon signed rank test. Friedman tests results are also shown under each column header.

	Start Error $\left(\chi^2_3 = 19.6, p < .001\right)$			End Error $\left(\chi^2_3 = 16.9, p < .001\right)$			Total Error $\left(\chi^2_3 = 22.4, p < .001\right)$			Length Error $\left(\chi^2_3 = 27.8, p < .001\right)$		
Pair	$W+$	p	r	$W+$	p	r	$W+$	p	r	$W+$	p	r
Machete - Window	2	$< .05$	0.9 L	0	$< .05$	1.0 L	0	$< .05$	1.0 L	0	$< .05$	1.0 L
Machete - CDP	55	$< .05$	1.0 L	32	0.70	—	55	$< .05$	1.0 L	55	$< .05$	1.0 L
Machete - Energy	6	0.05	—	14	0.39	—	0	$< .05$	1.0 L	51	$< .05$	0.9 L
Window - CDP	53	$< .05$	0.9 L	1	$< .05$	1.0 L	22	0.62	—	54	$< .05$	1.0 L
Window - Energy	15	0.23	—	0	$< .05$	1.0 L	4	$< .05$	0.9 L	0	$< .05$	1.0 L
CDP - Energy	53	$< .05$	0.9 L	12	0.39	—	51	$< .05$	0.9 L	55	$< .05$	1.0 L

average start, end, and total segmentation errors, and on all measures Machete was significantly different from window-based segmentation. Although energy based segmentation performed better on arc length error, we note that both Machete and energy perform well, where both are better than 6% (the lowest averages for all datasets). On the other hand, window-based segmentation performed at its worst, which may be due to the increased temporal variability in this dataset.

7.6 Machete in Practice

To better understand the viability of Machete in practice, we implemented the The Dollar General custom gesture recognition pipeline without automatic rejection threshold selection. Because we

are primarily concerned with the effect of segmentation on recognition and latency, we used a grid search to find optimal rejection thresholds for each condition discussed below.

Based on results reported in the prior section, we further decided to evaluate both Machete and Window based segmentation on Kinect input. Both techniques are comparable in start, end, and path length error, showing no statistically significant difference in accuracy when one selects an appropriate window size. Insufficient accuracy prohibits the use of CDP, which also impacts energy-based segmentation, as does computational performance. Further, since windowing is a popular technique, it is useful to understand where Machete stands with respect to window-based segmentation.

One challenge with windowing is that one must select an appropriate window size. Too short and gesture candidates will be truncated. Too large and the window will contain data of temporally adjacent activities. In both case, a recognizer may reject the result. In our prior test, we set the window size to that of its associated training sample length to understand the potential of window-based segmentation. In a practical application, one must balance accuracy with latency and use only a minimum number of windows. When gesture lengths are homogeneous, one can use a single window and expect reasonable segmentation. In our case, gesture length varies between short and long gestures; so it is unclear what window size we should use. For this reason, we evaluate five options: a single window using the (1) minimum, (2) middle, and (3) maximum sample length; (4) two windows using the minimum and maximum training sample length; and (5) all three together.

Since our goal with this evaluation is to understand how segmentation may impact user experience, we measure recognition accuracy and latency. See Chapter 2 for a discussion on the importance of these measures. Latency results were collected on an Intel Core i7-7700K with 24GB 2400MHz DRAM and an Nvidia GeForce GTX 1080, running Windows 10.

7.6.1 Recognition Accuracy

To measure the effect of segmentation on recognition accuracy in a realistic scenario, we use the following procedure: For a given participant P, we randomly select T training samples per gesture class. We train both the segmenter and Jackknife with said random sample, and then replay participant P's session through our pipeline, one frame at a time. During replay, we record all true positive tp, false positive fp, and false negative fn classification results—gesture candidates output by the segmenter that are not rejected by Jackknife. Results are then combined into an F_1-score, a commonly used accuracy measure balancing precision and recall:

$$F_1 = \frac{tp}{tp + \frac{1}{2}(fp + fn)}. \tag{7.34}$$

Recognizers achieve a perfect score when all performed gestures are correctly classified, but degrade as misses and misclassifications increase. In our evaluation, a true positive occurs when a participant performs the specified gesture and our pipeline correctly observes the correct pattern. Otherwise, missed patterns are treated as false negatives and missclassficiations as false positives. For each participant and level of T, we repeat the process ten times and average all F_1 score results into a single accuracy measure per individual. Training count T varies from 1 to 5.

Results are shown in Figure 7.14. We observe that poor window choices lead to poor recognition accuracy. Using only the maximum or mid window length, windowing achieves only a 61% and 74% accuracy, respectfully. By segmenting with the minimum length, accuracy increases to 88% for $T = 1$ and 89% for $T = 3$, but there is no further gain. Machete and multiple window segmentation significantly outperform single window segmentation, and are comparable to each other. Using two windows (minimum and maximum), accuracy starts at 92% and increases to 94% as T increases, and three window segmentation performs similarly. Machete-based segmentation also performs well, has the sharpest incline, and ends with the highest result, starting at 91% and

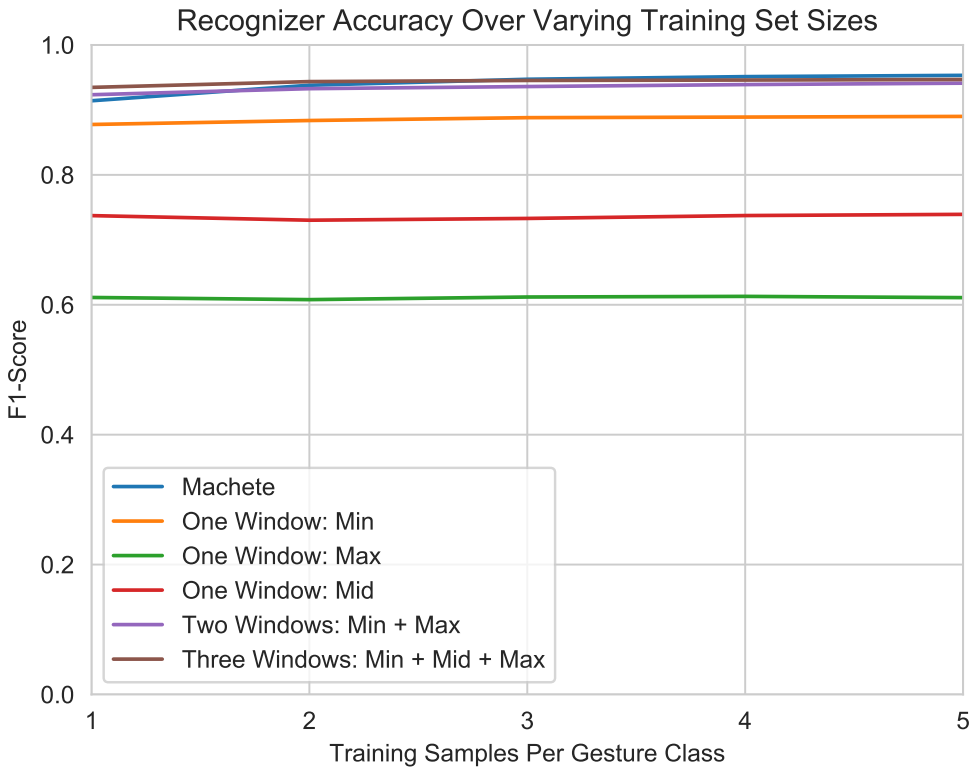


Figure 7.14: F_1-score accuracy for varying training set sizes for different segmentation strategies. Minimum refers to the length of the minimum length training sample, whereas maximum refers to the maximum length training sample and mid is their average.

ending at 95%.

As we saw earlier (Figure 7.13), a number of Kinect gestures share similar temporal variation, which is likely why the minimum sized window works best of all single window options. The remaining gestures are sufficiently long and captured by the maximum window. For gesture sets that possess greater variability than ours, selecting appropriate window lengths will prove to be more challenging. Further, we find that poor segmentation leads to suboptimal results, evidenced by a substantial reduction in recognition accuracies achieved by the single window methods. These lower results force one to use multiple windows, leading to observable latency degradations as we will soon see.

Figure 7.15: Screenshot of Sleepy Town [234] environment.

7.6.2 Latency

To evaluate the effect of segmentation on latency, we implemented our pipeline in a Unity based video game called Sleepy Town (see [234] for a full description). It presents a simple geometric 3D environment in which citizens wander between random points without purpose. We chose to use this simple platform because it uses many features common among modern games, including path planning, behavior scripts, collision detection and response, physics, scene management, and rendering. Our pipeline must, therefore, share system resources with other various ongoing processes. This means that gesture recognition overhead may directly impact frame rate and system responsiveness.

Due to COVID-19, we were unable to directly collect participant data. For this reason we decided to select an exemplar participant from our previous FTL data collection procedure and replay their session data through our pipeline as if it were read directly from a Kinect input device. We repeated

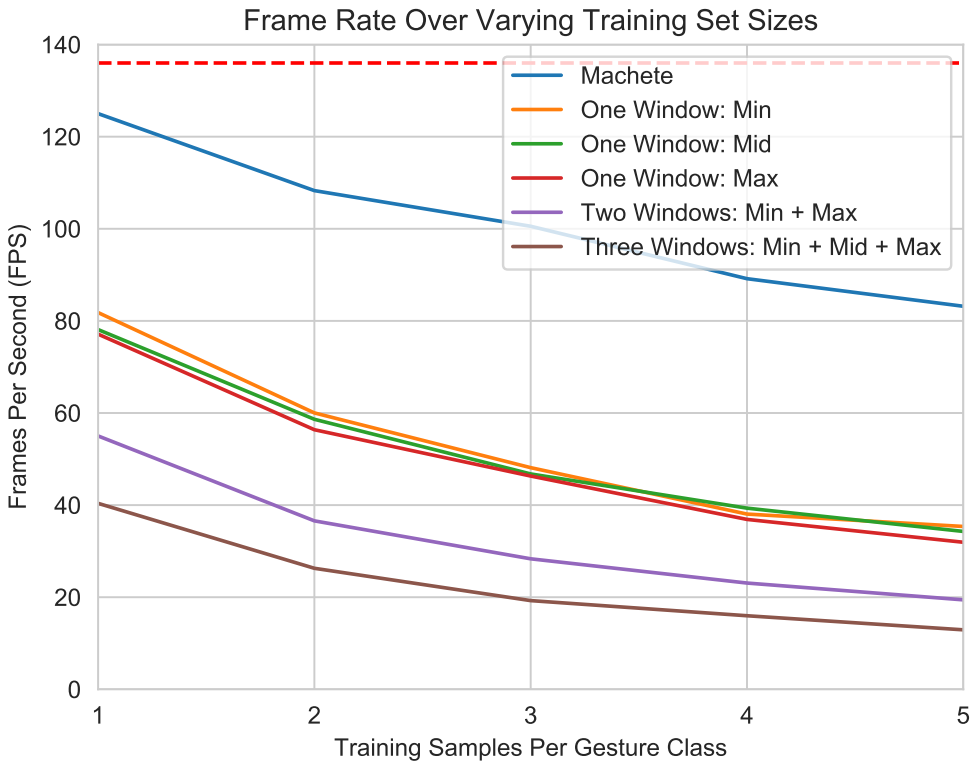


Figure 7.16: Frame rate over varying traing set sizes for the various segmentation configurations. Curves are in approximate order of the legend labels. The horitonal red dashed line is the frame rate of Sleepy Town without gesture recognition.

this procedure for varying training set sizes and segmenter approaches as discussed in the previous section. During this time, we tracked frame rate and recognition accuracies (which were confirmed to match our prior results).

Results are shown in Figure 7.16. Sleepy Town running without gesture recognition maintains approximately 137 frame per second (FPS). The single window segmenters drop down to near 60 FPS with one template per gesture class loaded, and performance continues to drop as more templates are loaded, ultimately hitting 32—35 FPS. Two window segmentation starts at 58 FPS and drops to 19 FPS. Three window segmentation drops from 40 to 13 FPS. Machete outperforms all window options, starting at close to maximum performance, 126 FPS, and dropping down to 83

FPS.

As expected, we find there is significant overhead when using window-based segmentation. Since recognition accuracy may be too low with one window in many cases, we expect one will prefer to use two windows, if not more (for instance, Multiwave [190] used five windows to handle temporal variability issues). We also note that all methods degrade as the number of templates increase, though Machete is the only method that continues to operate in an acceptable range with 85 templates loaded [42] (17 gestures $\times$ 5 per class). This increase in performance will benefit user experience in both desktop and virtual reality applications, where high latency can degrade performance and presence, as well as increase the probability of inducing motion sickness.

7.7 Discussion

Our experiments reveal that Machete is the only segmentation method that consistently achieves high accuracy across all datasets, and in most cases, Machete is the best performing method. Window-based segmentation is a close second on three datasets, but performed poorly on mouse data. Although there was not a statistically significant difference between Machete and windowing across all conditions, the direction of their differences remained constant. When we look at computational performance, only CDP can compete, but because of its poor accuracy, CDP is not a viable option. Machete's combined accuracy, speed, scalability, and simplicity makes it a clearly capable segmenter.

In more detail, we found that windowing works relatively well in the presence of minor temporal variabilities, but falters as variability increases. We can recover performance by employing multiple sliding windows, where we evaluate each window every frame and take the best result. A related practical issue with our evaluation is that we fit our window size to the training data, so

each template uses its own window. In practice, one may cluster templates into groups that share a common window size, or one might select just a few sizes that all templates share. The downside of using multiple windows is obvious though, more windows means more overhead means more processing, and there is still no guarantee a particular size will fit a given gesture instance. Alternatively, one can use a single long sliding window to segment input when and if gesture boundaries are delineated by periods of inactivity between actions. For instance, a two second gesture is adequately described by a four second window when a user holds their pose before and after gesticulation, and when the underlying recognizer is time invariant. This approach may be appropriate for low-activity interactions but not for high-activity interactions where users are engaged in constant motion. Machete does not suffer from these issues, which is why our technique outperformed window-based segmentation in every test.

We found that energy-based segmentation was often less accurate and its computational performance was also low. Although we were able to quickly calculate the input signal's energy profile, performance dropped because of an overabundance in minima, each of which resulted in multiple calls to the recognizer. We believe with more work we can develop a thresholding scheme that rejects certain minima and coalesces collocated minima to reduce the number of candidate gestures. However, based on work by Kahol *et al.* [116], we fear a workable solution will be complicated, device specific, or both. Interestingly, the method did very well on mouse data. There are often clear low-energy start and stop points in this dataset where the segmenter could latch, which is why accuracy is high in this case. In a related way, this also may be why the segmenter is less successful on full body gestures. Recall that gestures comprise pre-stroke, nucleus, and post-stroke activities [273], where the nucleus defines the main action. It may be that the nucleus is clearly delineated in mouse data but not in full body gestures. Regardless, because of the segmenter's inconsistent performance and high overhead, we believe it is not a good general first-choice technique for custom gesture recognition.

CDP was the worst performing segmenter in our evaluation. We noticed that CDP had consistently high start errors, and visual inspection of the associated distributions reveals a bias toward late segmentation. This issue is likely because of CDP's zero sink node cost, an issue we addressed in Machete. With respect to computational performance, CDP did very well, though because we were unable to find a reliable rejection threshold, it did not perform as well as Machete. This difference in part is due to Machete's improved local cost function. In other words, our modifications to CDP have a profound impact on performance.

Another issue with both windowing and energy-based segmentation is that these methods do not prune gesture candidates. The latter technique will identify potential gesture boundaries and invoke the recognizer with a query that it must score against each template. Other than duration and application-specific context, there is no a-priori information available that the recognizer can use to discard unlikely matches. Windowing has a similar problem. Whether templates share a window or are independent, the recognizer must evaluate each per frame. Another advantage of Machete is that the recognizer only evaluates one query-template pair per invocation, which further reduces the workload.

Advantages of Machete over other segmenters were also found in our recognition and latency evaluations. As expected, with better segmentation, we find gesture recognition accuracy improves. Otherwise, gesture sequences that are truncated or extended beyond their natural boundaries may be rejected by the underlying recognizer. In our case, to match Machete's accuracy, we were required to use two windows, but not without sacrificing significant computational resources. Frame rates in our Sleepy Town video game fell to below 20 FPS with two window segmentation as we scaled the training set size up from one to five samples per gesture class (85 samples total). In the worst case, Machete remained above 80 FPS. Low frame rates are known to impact perceived quality and performance in video games [42], which made windowing a suboptimal option. Informally, when running with window-based segmentation, we were also able to observe respon-

siveness degradation. Moreover, we had observed this issue in prior research and it became part of what motivated us to develop Machete.

We also note that performance can be impacted by gesture class similarity, where similar gestures may produce more candidates relative to when there is greater separation between gesture classes. Our Kinect dataset has several gestures that are related such as the left hook, left uppercut, left-right-left combo, push, and backflip, as well as the right handed variants. All of these require significant forward left arm movement. Yet, despite their similiarity, we find that Machete maintains a high level of performance throughout all of our evaluations. For these reasons, we believe practitioners and designers engaged in continuous custom gesture recognition work will be able to take advantage of our segmenter to improve both recognition accuracy and latency.

7.7.1 Limitations and Future Work

Presently, Machete is unable to localize end points that occur within the middle of long straight lines. If one produces a right arrow with an exceptionally long shaft, Machete will incorrectly estimate its end points as being near deviations from the template. Often such deviations naturally occur on gesture boundaries, but not always. Our correction factors help localize end points relative to the associated start, but when this estimate is wrong, the error may propagate to the end point as well. One solution may be to include a post processing step that refines segmentation, but we leave this to future work. Another issue is that Machete is not orientation invariant, but we do not intend to address this limitation—because Machete is sufficiently fast, one can simply add rotated samples to the training set, or depending on the target environment, orientation can be handled elsewhere within the application.

Another important question is how can we improve Machete so that it does not depend on an underlying recognizer, because an obvious benefit in independence is that we reduce overall system

complexity. We found that our approach is very effective at segmenting continuous input, but its measurement strategy cannot sufficiently separate gesture classes, and this is why we require external validation. We believe the underlying issue relates to elasticity. Recognizers like Jackknife that work on segmented input are able to put sequences into direct correspondence over both space and time because they resample the input and limit warping, whereas Machete does neither. As part of our future work, we plan to address this situation, perhaps by using correction factors that evolve over time and help prevent pathological warping.

7.7.2 What's In a Name?

A machete is a coarse cutting tool often used in tropical settings to remove unwanted overgrowth and shrubbery. It requires little training to use: one simply swings forcefully at a target and is left with clean-cut greenery. In much the same way, our segmentation method requires little background knowledge or gesture recognition experience. A practitioner can simply swing Machete at their continuous data and be left with well segmented data that any high precision recognizer can thereafter evaluate.

7.8 Conclusion

We have presented Machete, a reliable and fast segmentation technique for custom gestures. Through an extensive evaluation involving difficult high-activity data across a variety of input devices, we found that Machete's accuracy is competitive with other commonly used approaches and often performs better. However, Machete especially shines with respect to performance. Because of Machete's discriminatory power and simple pruning rules, it reduces the computational burden by as much as 98%. Further, our technique is straightforward, being a CDP variant with several novel

improvements, which allows for quick integration and rapid prototyping. Though more importantly, because of its high accuracy, one can use Machete to support custom interface design for continuous input device types. For this reason, we include Machete in The Dollar General.

CHAPTER 8: GESTURE PATH STOCHASTIC RESAMPLING[1]

Computerized clinic
For superior cynics
Who dance to a synthetic band

In their own image
Their world is fashioned
No wonder they don't understand

Rush
Natural Science

One limitation of gesture customization is that only a few training samples are given from which a system must learn to recognize general patterns. We have seen in prior chapters that although custom gesture recognizers like Penny Pincher and Jackknife perform well with just one sample per gesture class, performance improves as the training set size increases. This is contrary to the idea of customization. We could ask users to provide us with more samples, but it remains uncertain how well a user interface would be received if it came with this burden. As such, we require and strive for high accuracy with little data, and techniques that help us achieve this goal are of high value. One approach that helps us overcome this issue is to employ synthetic data generation (SDG), whereby gesture variations synthesized from available data are used to train the recognizer [62]. Synthetic data can help both nearest neighbor and parametric recognizers achieve higher accuracy [29, 141, 212, 258].

In this chapter, we introduce a novel SDG method called Gesture Path Stochastic Resampling

[1]This chapter contains previously published material adapted from the following article: Taranta II, Eugene M., *et al.* "A rapid prototyping approach to synthetic data generation for improved 2D gesture recognition." Proceedings of the 29th Annual Symposium on User Interface Software and Technology. 2016. See Appendix C for associated copyright information.

(GPSR) that adheres to $-family principles and is designed for rapid prototyping. Through an extensive evaluation, we show that recognition accuracy improves when one uses GPSR to increase their training set size, and that GPSR outperforms alternative SDG techniques in both accuracy and computation. Although we focus first on 2D gestures, we move to higher dimensional gesture recognition in the next chapter where we use GPSR to learn rejection thresholds for continuous custom gesture recognition. In this way, GPSR becomes a critical component in The Dollar General pipeline.

8.1 Gesture Path Stochastic Resampling

The design of our synthetic data generation method is motivated by several crucial objectives. Foremost, as a rapid prototyping technique, the approach should be easily accessible to the average developer: understood with little effort and without expert knowledge, and consequently be fast to implement as well as easy to debug. Yet even with its reduced complexity, improvements in recognition accuracy must remain competitive as compared to other state-of-the-art SDG methods. For the sake of adoptability, the approach should utilize only spatial coordinates given that timing and pressure information may be unreliable or even unavailable, but more importantly, artificial gestures should be synthesized with minimal computational overhead. Further, synthetic gestures should have a realistic appearance, not only for display purposes, but because severely deformed synthetic samples may lead to poor recognizer performance. Finally, the method should fit orthogonally and be complementary to already available gesture recognition techniques so that existing machinery might be leveraged without significant modification. In our view, *gesture path stochastic resampling* (GPSR) satisfies these criteria.

Similar in nature to the $\Sigma\Lambda$ model [141], one may think of a gesture as being described by a canonical velocity profile that is based on an action plan defining the gesture's shape. To recreate

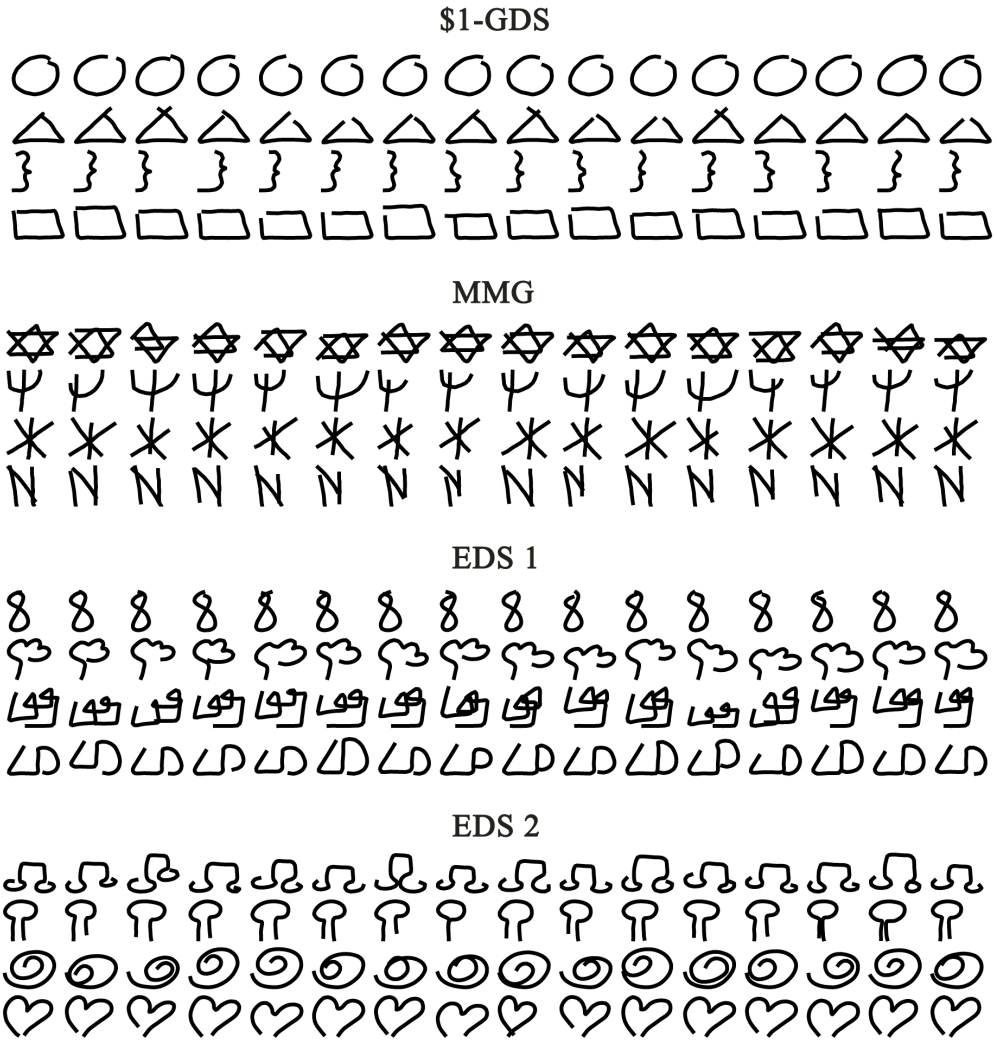


Figure 8.1: Example synthetic gestures from $1-GDS [267], MMG [9], EDS 1 [257], and EDS 2 [257]. The first column of each row is the original sample from which the remaining synthetic gestures are derived. All gestures are smoothed.

the gesture, a writer must execute the plan with a certain level of fidelity. Minor variability due to perturbations in the velocity profile will yield recognizable, yet uniquely different shapes, and so long as the writer's variation is reasonable, global characteristics of the shape will remain intact. However, rather than extract complex model parameters from a given sample and perturb the parameters post hoc, we can simulate reasonable perturbations on the sample directly.

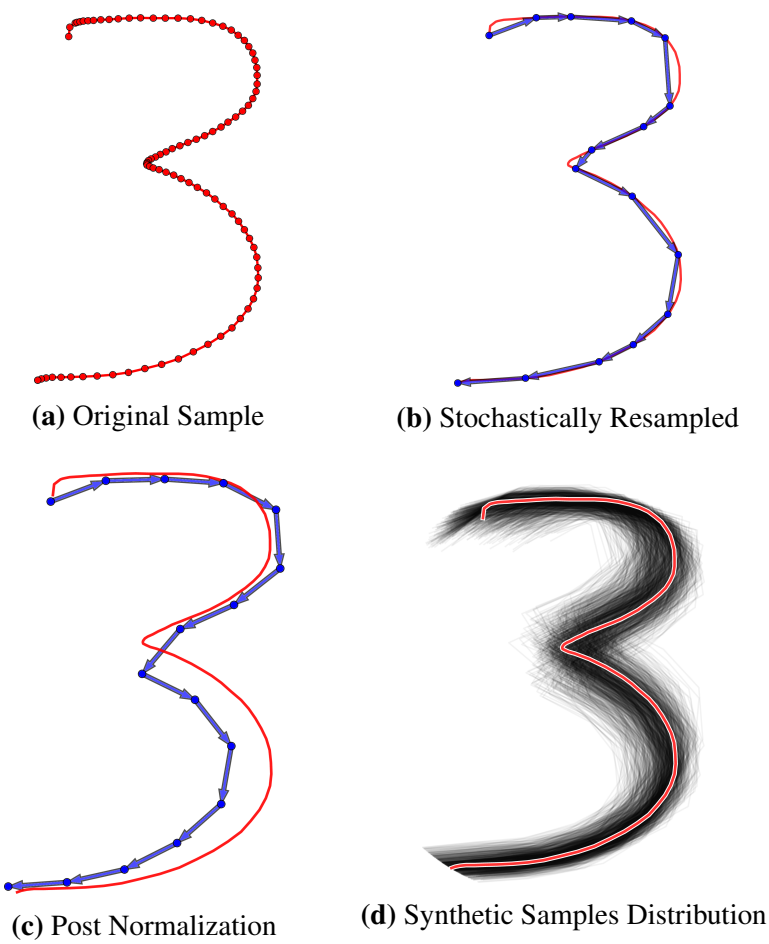


Figure 8.2: Illustration of the gesture path stochastic resampling process.

As a first step, we consider perturbations that result in the lengthening or shortening of gesture subpaths. To simulate such deviations from the action plan, consider resampling a stroke to N points so that the path length is *nonuniform* between points. Call any vector between two contiguous points along the gesture path an in-between point direction vector [238]. Now normalize all in-between point vectors to unit length and observe that for any two arbitrary vectors, the ratio of their lengths are altered as a result of the transformation. This approach is analogous to modifying the gesture path length between each pair of contiguous points, which is illustrated in Figure 8.3—some in-between point vectors will have been shortened while others will have been lengthened.

It is easy to see that each nonuniform resampling and subsequent normalization will lead to a different shape, and with repeated application, a distribution of synthetic gestures can be generated from a single sample.

Formally, let $\xi_1 = 0$ and $\xi_2, \ldots, \xi_N$ be a random sample of size $N-1$ from a uniform $\mathscr{U}(1, 1+\sqrt{12 * \sigma^2})$ population. Define an ordered list of gesture path ratios using the random sample:

$$r = \left(r_i = \frac{\sum_{j=1}^{i} \xi_j}{\sum \xi} \;\middle|\; i = 1 \ldots N \right), \tag{8.1}$$

so that $0 = r_1 < r_i < r_{i+1} < r_N = 1$. Further, a trajectory is defined as an ordered list of points $X = (\boldsymbol{x}_i \in \mathbb{R}^m \mid i = 1 \ldots N)$, $\mathscr{L}$ is the arc-length of the gesture path through all points from $\boldsymbol{x}_1$ to $\boldsymbol{x}_N$, and $\mathscr{L}(\boldsymbol{x}_i)$ is the arc-length distance to point $\boldsymbol{x}_i$ [251]. Similarly, denote $\mathscr{L}^{-1}(d)$ as the inverse arc-length function that returns the point $\boldsymbol{x}$ at distance $d \in [0,1]$ along the gesture path. Now define an ordered list of stochastic points using the ratios as follows:

$$Y = \left(\boldsymbol{y}_i = \mathscr{L}^{-1}\left(r_i \times \mathscr{L}\right) \;\middle|\; i = 1 \ldots N \right). \tag{8.2}$$

The in-between point vectors derived from the stochastic points are $\vec{Y} = (\vec{\boldsymbol{y}}_i = (\boldsymbol{y}_{i+1} - \boldsymbol{y}_i) \mid i = 1 \ldots N-1)$, from which a synthetic stroke is generated:

$$X' = \left(\boldsymbol{x}'_i = \boldsymbol{x}'_{i-1} + \frac{\vec{\boldsymbol{y}}_{i-1}}{\|\vec{\boldsymbol{y}}_{i-1}\|} \;\middle|\; i = 2 \ldots N \right), \tag{8.3}$$

where $\boldsymbol{x}'_1 = \boldsymbol{0}^m$. The synthetic stroke X' can then be scaled, translated, rotated, and smoothed as desired.

We chose to use a uniform random distribution after finding inconsequential differences between the uniform, normal, exponential, and beta distributions, which was contrary to the poor perform-

ing log-normal distribution. The lower bound of the uniform distribution was set to 1 only to avoid any probability of drawing 0 and the upper bound is a function of variance so that the spread of the distribution can optionally be tuned.

8.1.1 Removals

Another type of variation occurs when a writer skips over some detail of the action plan, such as when a gesture stroke is not fully articulated or when a corner is cut. To simulate this in a general way we introduce the removal count x that, when specified, indicates how many points from the stochastic stroke Y are randomly removed before generating the synthetic stroke X'. When removals are used, N should be adjusted to account for this reduction. Therefore, from hereon and for clarity, N refers to the length of the final synthetic stroke after the removal of x points, and implicitly the length of ξ, r, and Y become $N+x$.

8.1.2 Multi-trajectory Support

So far, we have described how to create single trajectory synthetic gestures. With only a few modifications, gesture path stochastic resampling can also work with multi-trajectory gestures. In a preprocessing step, first randomly permute the stroke set and randomly reverse a subset of the these strokes before combining them together into a single ordered list of points[2] X. In addition to coordinate data, each point also now possesses an integer trajectory ID. Stochastically resample X as before, while also interpolating the trajectory ID as one does with standard coordinates. Last, generate the synthetic sample, but break apart trajectory X' by discarding "over the air" points whose stroke IDs are between integer values. This results in a synthetic multi-trajectory gesture,

[2]This approach was inspired by $N's [8] handling of multistroke gestures.

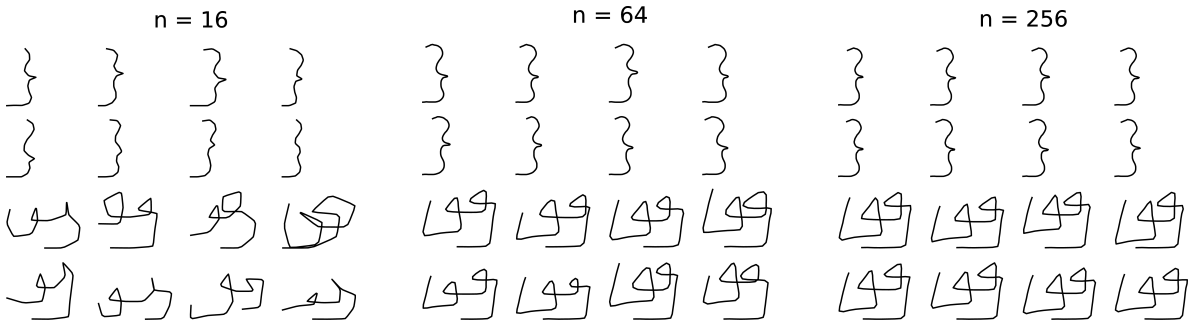


Figure 8.3: Effect of N on gestures of different complexity – left curly brace [267] (top) and triangle chain [257] (bottom)

and example results are shown in the second four rows of Figure 12.1.

8.2 Parameter Selection

Gesture path stochastic resampling requires the selection of three parameters: variance σ^2, removal count x, and resampling count N. With respect to variance σ^2, we found in early testing that this parameter had little influence on recognizer accuracy—any variance setting was sufficient to achieve good results. Therefore, we selected for $\sigma^2 = 0.25$ and held the parameter constant for the remainder of our analysis. Removal count x, on the other hand, had a noticeable impact on synthetic gesture quality. At $x = 2$, recognition accuracy results were indeed improved, but as x increased, gesture quality rapidly deteriorated. For this reason, we also decided to hold the removal count constant at two, and focus on the one parameter having the most significant impact on accuracy and gesture quality, the resampling count N.

As illustrated in Figure 8.3, one can see that the selection of N significantly impacts synthetic gesture generation. With a low resampling rate such as with $N = 16$, the left curly brace [267] has reasonable variation between samples, but most triangle-chain [257] samples are unrecognizable.

At $N = 64$ there is practically no variation between left curly brace samples, though triangle-chain samples are now improved and have a healthy variation. Finally, at $N = 256$, there appears to be almost no variation for either gesture. These observations motivate us to find a function of N based on properties of the gesture that yield reasonable results—a function that can analyze an individual gesture sample and select an optimal resampling count N to apply.

In order to decide on an optimal resampling strategy, we consider the effect of N on various geometric relative accuracy measures[3] [251]. Namely, we use these measures to analyze how well a population of synthetic gesture samples matches a population of human generated samples. To start, Vatavu *et al.* [251] define the shape error (ShE) of a candidate gesture to be a measure of deviation from the gesture class's canonical form:

$$\mathrm{ShE}(X) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{x}_{\sigma(i)} - \overline{\boldsymbol{x}}_i\|, \tag{8.4}$$

where $\overline{\boldsymbol{x}}$ is the canonical point series and $\sigma(i)$ is a permuter of the candidate gesture points, which is used to find an optimal alignment. Shape variability (ShV) measures the standard deviation of the individual shape errors. Thereafter, bending error (BE) is a measure of average deviation from the absolute curvature of the gesture class's canonical form:

$$\mathrm{BE}(X) = \frac{1}{n}\sum_{i=1}^{n-2} \left|\theta_{\sigma(i)} - \overline{\theta}_i\right|, \tag{8.5}$$

where θ_i is the turning angle between in-between point vectors i and $i+1$. Bending variability (BV) again is the variance of these individual errors.

Using a similar evaluation methodology to that described shortly, we calculated all four relative

[3]In addition to geometric measures, Vatavu *et al.* [251] also define a set of kinematic accuracy measures that we do not include in our analysis given that GPSR does not attempt to synthesize time stamps. For similar reasons, the articulation accuracy measures were also not studied.

Table 8.1: Datasets used in evaluations.

Name	Ref	Multistroke	Gestures	Participants
$1-GDS	[267]	No	16	10
EDS 1	[257]	No	18	14
EDS 2	[257]	No	20	11
LP Training	[238]	No	40	24
MMG	[9]	Yes	16	20

accuracy measures for $N \in \{8, 16, 32, 64, 128\}$. We found that the ShE and ShV were significantly correlated ($r(549) = .91, p < .0001$), as were BE and BEV ($r(549) = .80, p < .0001$). Given this strong correlation between the error and variance metrics, we decided to focus on ShE and BE only. Specifically, we were interested in the mean ShE and mean BE for a population of gesture samples. To differentiate between the real population and a synthetic population we refer to these means as the real ShE (real BE) and syn ShE (syn BE), respectively. Further, the *mean ShE percentage error* is given in Equation 8.6, where each summand compares the syn and real ShE for one gesture, which gives us an error term for that gesture. Thus the equation averages these errors together, and our goal is to minimize this overall error:

$$\text{ShE \% Err} = \frac{100}{G} \sum_{i=1}^{G} \frac{|\text{real ShE} - \text{syn ShE}|}{\text{real ShE}}, \tag{8.6}$$

where G is the number of gestures under consideration. The mean BE percentage error is defined similarly.

To carry out our investigation, we utilized the five datasets described in Table 11.1. All of these datasets have been widely used in various studies except LP Training [238]; we included this dataset only to help avoid overfitting given that it replicates a number of gestures found in the other datasets.

For a given gesture, we first scaled and aligned[4] all samples within the population, after which we followed the procedure described by Vatavu *et al.* [251] to select the average template. That is, we uniformly resampled all samples to a fixed number of points, calculated the mean of the population, and selected the sample closest to the mean to serve as the canonical (centroid) sample of the gesture class. However, because all strokes were first aligned, we choose our permuter $\sigma(i)$ to be the identity permutation. With the canonical gesture in hand, we found the optimal N that minimized the ShE percentage error[5]—for each value of N tested, we generated 512 synthetic samples from the centroid. With these samples, we then calculated the syn ShE for the population. Note that ShE was prioritized at this stage because the metric relies on point correspondences which directly, and sometimes indirectly, relate to the distance metric employed by various rapid prototyping gesture recognizers. Further, we also assumed that improvements in ShE error would lead to improvements in BE error.

Since our goal was to find a function of N based on properties of a given sample, we considered seven features derived from summaries provided by Blagojevic *et al.* [22]: curvature, absolute curvature, closedness, direction changes, two density variants, and stroke count. In addition to optimal N, we also extracted these features from the centroid sample. Using techniques from [134], we found that the optimal N had a logarithmic relationship with its predictors, which is why we first transformed the response. Thereafter, using stepwise linear regression, we identified those features that best explained the variability of the 110 gestures (see Table 11.1). In the end, we identified two features that together achieved high performance:

$$\text{closedness} = 1 - \frac{|| \boldsymbol{x}_n - \boldsymbol{x}_1 ||}{\text{diag}}, \tag{8.7}$$

[4]Alignment was decided by finding the per sample stroke order permutation and direction that minimized the non-GSS $1 [267] distance across the population.

[5]We set $N = 16$ as a lower bound, since lower values can result in poorly malformed gestures.

and,

$$\text{density} = \frac{\mathscr{L}}{\text{diag}}, \tag{8.8}$$

where diag is the gesture's bounding box diagonal length. Absolute curvature actually accounted for negligibly higher variability over density, but since the feature may be unreliable due to jitter or wobbling and because destiny is a simpler approach, we favored a parsimonious solution.

Now with good features selected, we performed multiple linear regression to find an equation for optimal N. A significant regression equation was found ($R^2 = .59$, $F(2,109) = 75.62$, $p < .0001$). The intercept ($p < .0001$), density ($p < .0007$) and closedness ($p < .0001$) parameters were significant, yielding the following equation:

$$N = e^{1.67 + 0.29\ \text{density} + 1.42\ \text{closedness}} \tag{8.9}$$

Further, the cross validated coefficient of determination[6] ($Q^2 = .56$) was approximately equal to $R^2 = .59$, which is another indication of a good fit. Residuals were also confirmed to be statistically normal using a Shapiro-Wilks test ($W = .99, p = .37$).

We consider $R^2 = .59$ to be a good result because there is a great deal of variability between datasets. For instance, some selected differences in real ShE are the heart gesture at .061 and .142; rectangle at .045 and .054; five point star at .099 and .135; and triangle at .056 and .081. These differences are likely related to how the datasets were collected, including the device, instructions, and software used. By applying the optimal N equation to each of the 110 gesture centroids from Table 11.1, we find that the N values range from 16 to 69, and have a mean of 31 (SD=13.1).

[6] $Q^2 = 1 - \frac{PRESS}{TSS}$.

Table 8.2: Mean gesture recognition percentage error (and SD) over all template matching recognizers for one and two training samples per gesture, from which 64 gestures are synthesized per training sample (see Evaluation) on $1-GDS [267], MMG [9], EDS 1 [257], and EDS 2 [257], as well as the ShE and BE percentage errors. Note that the optimal N value ranges from 16 to 69 depending on the gesture's centroid.

N	**Rec % Error**	**ShE % Err**	**BE % Err**
Optimal	3.47 (3.81)	26.06 (21.24)	21.15 (15.23)
8	—	1137.72 (9538.78)	95.86 (85.58)
16	4.68 (5.05)	80.22 (67.82)	33.37 (37.34)
32	3.80 (3.98)	38.85 (25.14)	25.27 (17.28)
64	4.20 (4.18)	44.44 (24.04)	33.63 (14.18)

8.2.1 Evaluation of Optimal N

To understand if optimal N (Equation 8.9) is effective at simulating a realistic distribution, we calculated the relative metrics over varying $N \in \{8, 16, 32, 64\}$ and optimal N. Results can be found in Table 8.2. Overall, optimal N had the lowest ShE percentage error ($M = 26\%, SD = 21\%$) and, as compared to its runner up $N = 32$ ($M = 39\%, SD = 25\%$), the result was statistically significant based on a Wilcoxon signed-ranks test ($p < .0001$). Similarly, optimal N also had the lowest BE percentage error ($M = 21\%$, $SD = 15\%$) that again, compared to $N = 32$ ($M = 25\%, SD = 17\%$), was significant ($p < .004$).

Further, to ensure our approach did not degrade recognizer performance (for instance, by reducing or over inflating population variance as compared to other static values of N), we also evaluated recognition accuracy for each level. Optimal N achieved the lowest mean error 3.47% ($SD = 3.81$) for one and two training samples expanded into 64 synthetic gestures per sample, using unistroke recognizers on unistroke datasets and multistroke recognizers on MMG [8]. The second closest was $N = 32$, having a mean error of 3.80% ($SD = 3.98$); although, the difference between levels

was not significant ($F(3,220) = .41, n.s.$).

As a result of this analysis, we determined that optimal N (which is unique per sample) is able to synthesize a gesture population more precisely than any static value of N. This is highly desirable, because for unknown datasets, we need to reduce the probability of generating unrealistic samples that cross the decision boundary between gestures classes, since malformed gestures have the potential to ruin recognizer performance. On the other hand, we also do not want to restrict synthetic data generation so much so that there is insufficient variation to be of any use to a gesture recognizer, and therefore optimal N is utilized to strike a balance between these objectives. For this reason, optimal N was used throughout the remainder of our evaluations.

8.3 Evaluation: Recognition Accuracy

Our evaluation strategy was similar to that used by Wobbrock *et al.* [267]. One key difference in our protocol is that instead of performing writer-dependent testing, we use a mixed-writer protocol. Given a particular dataset comprised of G gestures, all samples from all participants are pooled together. Then for each gesture class, T real samples are selected from the pool and 1 remaining, independent sample is selected for testing. If synthetic data generation is being used, S synthetic samples per real sample are generated for training, which means the recognizer is trained with $G*T*S$ samples. Once the recognizer is trained, each sample in the validation test set is classified, which results in G recognition tests. Note that the validation set comprised only real samples, not synthetic samples. These G results are subsequently combined into a single average error rate. This procedure is repeated 1000 times and the result from each iteration is further averaged into a single overall recognition error rate.

In this evaluation, the levels of T are increased from 1 to 5, and the levels of S are $\{8, 16, 32, 64\}$.

We note that it may seem biased to test a recognizer trained with T real samples to a recognizer trained with $T * S$ synthetic samples, but it is important to note that in both cases, exactly the same number of real samples are provided. The key difference is in what advantage SDG can provide with respect to the recognition error rate. One reason why we train with more synthetic samples is because there are a lot of redundancies in the synthetic sample distribution, where some synthetic samples provide no additional value, which is discussed further in the limitations section.

All datasets appearing in Table 11.1 except LP Training were utilized in our recognition accuracy tests, as these are familiar datasets commonly appearing in the literature. We consider the problem described in the previous section of finding an optimal N function that produces realistic distributions to be different from that of finding a suitable N that results in high recognition accuracy. However, it is important to note that since we leverage the same datasets, it is possible that optimal N is overfit in these tests. Static values $N = 16, 32$, and 64 were also evaluated, and as before we found that $N = 32$ achieves the lowest average error rate of all static values; and their is no statistical difference between optimal N and $N = 32$. Thus, we do not believe there was overfitting in this case. For this reason, we report only optimal N in this section. Further, since we are primarily concerned with recognition accuracy when only one or two samples are given per gesture, we restrict our formal analysis to $T \in [1,2]$, and the remaining levels are only used to show that the trends continue as T increases, as is shown in Figure 8.4. As a last note, since error rates tend to be skewed towards zero and violate ANOVA assumptions, we used the Aligned Rank Transform (ART) method [265] to carry out our analysis.

8.3.1 SDG Methods

In addition to running all recognizers without synthetic data generation (the baseline), three SDG methods were evaluated: GPSR, Perlin noise, and $\Sigma\Lambda$. GPSR was implemented as discussed

where samples are stochastically resampled according optimal N (Equation 8.9). We used the Perlin noise implementation developed for [49]. This implementation includes several parameters that influence the shape of the generated synthetic sample (such as map resolution, amount of noise, *etc.*). Suitable values for these parameters were established via personal contact with the authors. Moreover, the authors expressed that the noise map was further smoothed via Gaussian blur prior to application, and these considerations are incorporated in our implementation. It is worth mentioning Perlin noise map generation is time consuming, which makes synthetic sample generation slow as compared to GPSR or $\Sigma\Lambda$ (post parameter extraction). To reduce the time needed to run our experiments, we precomputed 2048 Perlin noise maps and cached them to disk prior to application. Although considerable speed improvements were observed, these cached maps required 64 MiB of storage space.

Our $\Sigma\Lambda$ implementation is based on the recent parameter extraction algorithms described by Martín-Albo *et al.* [160]. Since parameter extraction can also be a time consuming process, we first extracted the $\Sigma\Lambda$ models for all samples in the aforementioned datasets and used only sufficiently high quality models in our evaluation. That is, per [141], we required that the signal to noise ratio of a reconstructed model be 15dB or greater; otherwise the sample was excluded. Since the success of parameter extraction is dependent on signal quality, low resolution strokes can lead to difficult situations. One example problematic scenario occurs with the MMG [8] dataset where multiple points have the same timestamp, which causes an incorrect velocity calculation. We addressed these issues as much as possible, but it should be noted that our implementation is unlikely to be as robust as alternative propriety implementations. This is why we consider our results to be a reasonable lower bound on what is possible with the $\Sigma\Lambda$ SDG method.

8.3.2 Recognizers

8.3.2.1 Template Matching Recognizers

Since gesture path stochastic resampling is designed to be used as a rapid prototyping technique, we expect $-family and other related recognizers to be paired with our method. Therefore, we decided to evaluate GPSR with six such recognizers: $1 [267], Protractor [143], $N [8], $N-protractor [9], $P [250], and Penny Pincher [238]. An additional benefit is that a variety of distance metrics are used throughout this suite of recognizers; so although these methods are all template matching, a large variety of techniques are represented.

8.3.2.2 Parametric Recognizers

Two parametric recognizers, Rubine's linear classifier [215] and naive Bayes were implemented and trained using the same set of features. We found that the original set of 13 features discussed in [215] were inadequate for mixed-writer gesture recognition, and some features were unusable, since GPSR does simulate timestamps for example. To overcome these issues, we included some of the most prominent features described in [22]. The final features we selected were the cosine ([215]:1)[7] and the sine ([215]:2) of the initial angle, angle of the bounding box's diagonal ([215]:4), the distance between endpoints ([215]:5), cosine ([215]:6) and the sine ([215]:7) of the angle between the first and last point, aspect ([22]:7-2), total angle traversed ([215]:9) as well as some convex hull related features such as length:perimeter ratio ([22]:2-6), perimeter efficiency ([22]:7-16) and perimeter:area ([22]:7-17) ratio.

[7]This notation signifies the feature number as presented in the paper referenced.

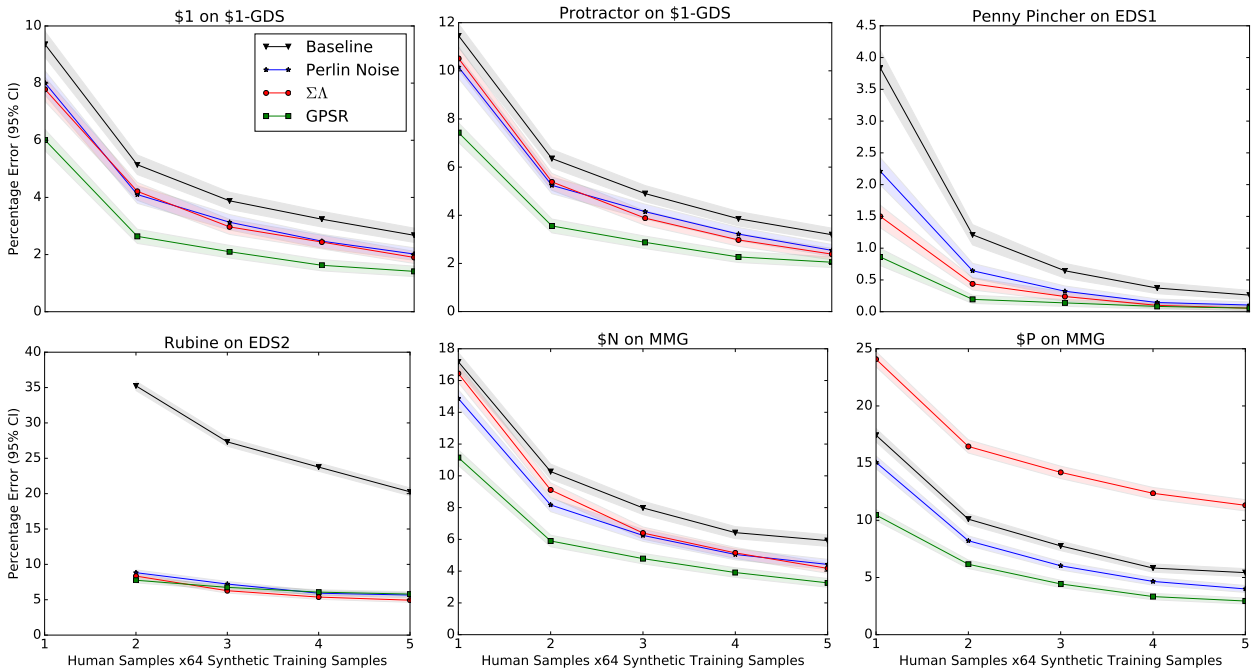


Figure 8.4: Accuracy results for various configurations. In each graph, the horizontal axis is the number of human samples per gesture used for training, where $S = 64$ synthetic samples were created per real sample. Results were randomly selected so as not to highlight any one particular recognizer and dataset. However, across the board, one will notice that mean recognition errors are significantly reduced using GPSR with gestures being stochastically resampled to optimal N.

8.3.3 Recognition Errors (Accuracy)

Figure 8.4 shows results for various recognizers on different datasets. These results were selected so as not to highlight any particular recognizer, dataset, or SDG method, though the results are consistent across all tested scenarios. One exception is naive Bayes (discussed below). Further, the reader may notice that ΣΛ performance is below baseline performance on the $P MMG dataset, but this result is compatible with those reported in [141]. Table 11.2 gives detailed recognition error rates for the best performing recognizer for each dataset given one real training sample per gesture. GPSR in all cases achieves the best performance. Minimally with 8 synthetic samples per training sample loaded, GPSR reduces the error rate by 18% on MMG and 30% on $1-GDS, whereas with

Table 8.3: Recognizer percentage error rates (SD) and their associated percentage error rate reductions from baseline (without SDG) given one real training sample per gesture ($T = 1$), comparing gesture path stochastic resampling (GPSR), ΣΛ and Perlin noise (PN) for $S = 8$ synthetic samples per real gesture and $S = 64$ across four datasets. The gesture recognizer shown is the one that achieved the lowest error rate for the given dataset and S. In all cases, GPSR achieves the best performance.

	$1-GDS [267]				**EDS 1 [257]**			
	Pincher, $S = 8$		Pincher, $S = 64$		$1, $S = 8$		$1, $S = 64$	
	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%
None	8.59 (6.56)	—	8.59 (6.56)	—	2.16 (3.29)	—	2.16 (3.29)	—
GPSR	6.05 (5.61)	30	4.76 (5.08)	45	1.11 (2.39)	49	0.46 (1.60)	79
ΣΛ	7.32 (5.98)	15	6.41 (5.71)	25	1.92 (3.21)	11	1.38 (2.78)	36
PN	7.34 (6.27)	15	6.89 (6.02)	20	1.67 (2.96)	22	1.31 (2.64)	39

	EDS 2 [257]				**MMG [8]**			
	Pincher, $S = 8$		Pincher, $S = 64$		$P, $S = 8$		$P, $S = 64$	
	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%
None	2.13 (3.09)	—	2.13 (3.09)	—	17.4 (9.20)	—	17.4 (9.20)	—
GPSR	1.11 (2.33)	48	0.59 (1.64)	72	14.3 (8.21)	18	10.5 (7.54)	40
ΣΛ	1.37 (2.56)	36	0.92 (2.10)	57	26.6 (9.96)	-49	24.1 (9.59)	-38
PN	1.58 (2.67)	26	1.24 (2.39)	42	15.0 (8.79)	09	15.1 (8.79)	14

the other two datasets, improvements approach 50%. At 64 synthetic samples per gesture, $P on MMG sees a 40% improvement, and $1 on EDS 1 enjoys a 79% reduction in the error rate. ΣΛ and Perlin noise also see improvements, but to lesser extent as can be seen in the table. Since we ran a vast number of tests, in what follows, we discuss average results across all recognizers and datasets. However, the full set of results can be found on our website.

8.3.3.1 Template Matching Gesture Recognizers

Unistroke Gestures. Compared to the baseline percentage error ($M = 5.49, SD = 4.03$), without SDG, all methods showed an improvement in accuracy. GPSR achieved the lowest error ($M = 3.10, SD = 3.04$), which was followed by ΣΛ ($M = 4.33, SD = 3.72$) and Perlin Noise ($M =$

$4.32, SD = 3.51$). These differences were statistically significant ($F(3,464) = 8.05, p < .0001$), and a post hoc analysis using Tukey's HSD indicated that GPSR is significantly different from all other methods ($p < .005$), although baseline, $\Sigma\Lambda$, and Perlin noise were not significantly different from each other.

Multistroke Gestures. With the MMG [9] dataset, results were similar. GPSR ($M = 10.10$, $SD = 3.29$) achieves the highest performance, compared to baseline ($M = 14.67$, $SD = 4.42$), which was followed by Perlin noise ($M = 12.68$, $SD = 3.86$), and $\Sigma\Lambda$ ($M = 16.98$, $SD = 5.25$). Again, these differences were significant ($F(3,74) = 11.25, p < .0001$), and a post hoc analysis showed that the SDG methods were significantly different from the baseline ($p < .0002$). However, not all SDG methods were not significantly different from one another. GPSR was significantly different from the baseline ($p < .04$), whereas Perlin noise and $\Sigma\Lambda$ were not significantly different from the baseline.

8.3.3.2 Parametric Gesture Recognizers

Both parametric recognizers were substantially improved by all SDG methods. The best performing method was Perlin noise ($M = 13.27$, $SD = 6.52$), which was very closely followed by GPSR ($M = 13.75$, $SD = 6.10$). $\Sigma\Lambda$ ($M = 15.57$, $SD = 7.01$) was also well below the baseline ($M = 32.96$, $SD = 9.47$). These results were statistically significant ($F(3,152) = 10.998, p < .0001$), and again only GPSR was significantly different from the baseline ($p < .0001$).

Upon further inspection, we found that with Rubine, GPSR achieved the lowest mean error ($M = 11.46$, $SD = 5.18$), which was followed by Perlin Noise ($M = 13.42$, $SD = 5.85$). Conversely, with naive Bayes, Perlin noise achieved the lowest mean error ($M = 13.12$, $SD = 7.25$), followed by GPSR ($M = 16.04$, SD=6.18). Naive Bayes appears to be the only case where Perlin noise achieved a better result than GPSR.

8.4 Evaluation: Runtime Performance

To determine the average time required to generate one synthetic sample, we ran additional tests specifically to measure synthetic gesture generation speed with the $1-GDS and MMG data sets. The performance of Perlin noise was evaluated both with and without using cached maps. The tests were performed on a Surface Pro™ 3 featuring a low-power dual core Intel Core-i7 mobile processor running at 2.30 GHz and 8 GiB of RAM. Table 8.4 summarizes the results.

Table 8.4: Average time required to generate one synthetic sample by each SDG method. All values are reported in microseconds and are averaged over 256000 trials. Perlin† is performed using cached Perlin maps, and the ΣΛ time does not include parameter extraction, which can take several seconds.

	$1-GDS		**MMG**	
	M (μs)	SD	M (μs)	SD
GPSR	6.75	0.43	8.34	0.45
ΣΛ	125.46	14.96	129.32	12.09
Perlin	1101.16	79.43	1091.89	30.31
Perlin†	6.82	0.49	7.38	0.64

According to table 4, the only method that is marginally faster than GPSR is cached Perlin noise. However, this superiority comes at the cost of additional storage needs. As mentioned before, caching 2048 Perlin noise maps requires 64 MiB of storage which may constrain its use on devices where available memory for applications is limited to a few hundred megabytes. Without caching, Perlin noise was the slowest method tested. Further, it is evident that all methods performed slightly worse when generating synthetic multistroke samples, which coincides with expectations.

8.5 Discussion

In all objectives set forth, gesture path stochastic resampling succeeds. Foremost, we believe the technique is rapid prototyping appropriate. For example, GPSR is a straightforward generalization of uniform resampling, i.e., with $\sigma^2 = 0, x = 0$, and the multistroke extension utilizes concepts from \$N [8] and \$P [250] without being more complex than either. Despite GPSR's reduced complexity, as compared to other state-of-the-art SDG methods, we see from the evaluation that our method is very competitive, for example, achieving a 79% error rate reduction over the EDS 1 [257]. Indeed the accuracy of all recognizers are improved and, in most cases, GPSR is the best performing method. With respect to synthetic gesture realism, using optimal N, GPSR achieves a low ShE and BE percentage error for populations of synthetic gestures generated around centroid samples. Finally, in terms of computational complexity, we see that if noise maps are precomputed, then cached Perlin noise is slightly faster than GPSR; otherwise, GPSR is able to synthesize samples much faster than the other methods, and so our approach can be used in realtime to generate potentially hundreds of synthetic samples.

We see that as the number of real training samples increases, GPSR continues to show significant improvements in the error rates, though we can also observe that accuracy is converging between recognizers trained with and without synthetic data, see Figure 8.4. The biggest difference in performance occurs when only one or two real samples are available, which is our primary research and is especially important for gesture customization such as for gesture shortcuts [10]. As Appert and Zhai remark: "Users tend to be reluctant to invest time and effort upfront to train or adjust software before using it." This sentiment is echoed by Li [143] who notes that users are unwilling to provide more than a small set of samples for training. Consequently, methods that achieve more with less are of high value in gesture recognition. Still, even in situations where a user can provide numerous samples, the increase in accuracy was substantial for all levels of real training samples

with the tested parametric recognizers. This result shows that for custom gestures, even when many samples are provided, synthetic data generation remains quite useful.

It is interesting to compare uniform and stochastic resampling with respect to the resampling rate N. Vatavu [245] was able to show that template matching recognizers employing a Euclidean or angular distance metric are able to achieve high recognition rates with as little as $N = 6$. High accuracy is possible because corresponding points along the gesture path of two samples from the same class are equivalent (the distance between points in the feature space is small), which is generally untrue for samples from different gesture classes. With stochastic resampling, on the other hand, small values of N dramatically shifts points along the gesture path, which has potential to move points out of correspondence and this helps to explain why, in part, GPSR is useful in generating new variations. Unlike uniform resampling, however, we found that GPSR depends on a unique N per gesture to achieve a reasonable ShE. If realism is unimportant, static $N = 32$ is also a good compromise where smaller values result in too much variability and larger values, not enough.

8.5.1 Limitations

Although we strived to develop a general approach that works well for most situations, there are still a number of limitations one should bear in mind before choosing to use our method. First, like many synthetic data generation methods, GPSR does not synthesize timestamps, which is an important part of some gesture recognition strategies, *e.g.*, for features based on velocity and duration [215] or stroke segmentation [97]. Timestamps from an original sample can be interpolated and applied to synthetic samples, but this naïve approach may not produce realistic results. An alternative approach may use the findings of Cao and Zhai [30] in modeling gesture stroke production times based on characteristics of a given gesture, but we leave this for future work.

Another limitation of GPSR is that synthetic samples are bound to the seed samples from which they are derived, and so our method may not sufficiently capture form variability, especially between writers. If there are valid alternative productions of a gesture, such as skewed edges on a right square bracket, GPSR will not generate these forms given that we modify the subpath *length* between points, not the subpath *direction*. Additional geometric transformations like shearing, scaling, and bending, applied globally or to a subpath may help overcome this issue, an approach that has been successfully used to generate synthetic textlines [242].

Since synthetic samples do not deviate substantially from the seeds, GPSR generates a number of samples that appear redundant in the recognition space—samples do not improve recognition accuracy because of their similarity with the seed and other synthetic samples. This limitation leads to two issues: we need to generate a large distribution of synthetic samples to achieve good coverage and these redundant samples may cause overfitting in a parametric recognizer. Template matching recognizers will be impacted by the distribution size since recognition time increases linearly with the number of templates. Penny Pincher is designed specifically for speed and large template counts, but practitioners may need to be careful to balance the training set size with application and domain specific requirements when using other recognizers. As part of our future work, we intend to investigate how to cull a synthetic sample distribution down to just a few good samples. One option we discuss in Chapter 11 is to use random mutation hill climb to aid in template selection for $-family recognizers, and they were able to achieve high accuracy with significantly fewer templates, but the process is offline. We prefer an online process in which samples can be discarded immediately based on those samples already synthesized.

Also not addressed in this work is how to handle small strokes in a multistroke gesture. For instance, the dot in an ‘i’ or ‘!’ may be discarded with a nonzero removal count $x > 0$; or if a small stroke appears in the middle of a gesture, the resampling algorithm may not produce any points

within the associated subpath[8]. However, until a more elegant solution is identified, logic can be added to ensure small strokes are not skipped.

8.5.2 Future Work

Although we set out to design a rapid prototyping technique to complement such gesture recognizers, we discovered that GPSR is quiet capable; and so to better understand the limitations of gesture path stochastic resampling and to understand if GPSR is appropriate for general use, further studies utilizing additional recognizers and SDG methods are warranted. Since we evaluated recognition accuracies in a mixed-writer scenario, we also require additional testing for pure writer-dependent and writer-independent scenarios. Our hope is that our method can help further bridge the gap between robust, high quality, commercial grade gesture recognition and rapid prototyping recognizers. For instance, it is desirable to evaluate the performance of our proposed approach when used for training support vector machines or random forests. Conversely, we also wish to evaluate if GPSR is suitable for testing recognizers, rather than training only.

We also plan to study user perception on synthetic gestures. A preliminary study in which participants rated the realism of synthetic and real gestures revealed significant differences between GPSR, $\Sigma\Lambda$, and Perlin noise samples, but not between GPSR and real samples. These results show promise that GPSR may be an appropriate method for rendering synthetic gestures that look real, but more work is needed to test and validate these results. Finally, beyond 2D gestures, we found that GPSR works reasonably well with signatures, sentences, and illustrations (see Figure 8.5 for an example), though refinement is certainly necessary. For instance, we found that optimal N was unsuitable for such complex structures and, for instance, sentences will often run off their baseline. Finally, we would also like to explore how GPSR can be applied towards 3D gesture recognition.

[8]This is also true for uniform resampling.

Figure 8.5: Stylistic sketch created by combining edge detection with GPSR.

8.6 Conclusion

In this chapter we presented a novel technique for synthetic data generation that significantly improves gesture recognition accuracy. Through an extensive evaluation we demonstrated that GPSR is highly competitive with current state-of-the-art SDG techniques, achieving as much as a 70% reduction in recognition error rates. Using optimal N, GPSR is also able to generate synthetic gesture distributions that approximate real distributions based on the ShE and BE relative accuracy measures. Further, due to its minimalistic design and low computational cost, GPSR is also applicable to rapid prototyping. In the next chapter, we will extend GPSR to support 3D data and show how it can be used to help decide a rejection threshold for identifying non-gesture data patterns.

CHAPTER 9: THE VOIGHT-KAMPFF MACHINE

TYRELL: Is this to be an empathy test? Capillary dilation of the so-called blush response, fluctuation of the pupil, involuntary dilation of the iris.

DECKARD: We call it Voight-Kampff for short.

Vangelis
Blush Response

Our goal with this work is to deliver a complete pipeline for continuous custom gesture recognition that is accurate yet accessible. We have already made significant progress toward this goal with Jackknife (Chapter 6) for device-agnostic recognition and Machete (Chapter 7) for continuous segmentation. These techniques are together able to isolate candidate gestures in a continuous stream and reliably measure their dissimilarity relative to exemplar template gestures. However, a critical piece of our pipeline that remains incomplete is that of a proper rejection criteria. When Machete submits a candidate gesture to Jackknife, our system must first decide if its similarity to the associated class model is sufficiently high, and thereafter determine if the system should be notified that a gesture has been recognized. The difficulty in making this decision manifests when classes are close together in their feature space, the system favors recall over precision, or a combination thereof. While accept-reject logic is itself application-dependent, and a practitioner may resolve these issues at the application level, one will still require an automatic rejection threshold selection method to avoid error-prone trial and error.

In greater detail, a **rejection threshold** defines the maximum dissimilarity measure by which one may claim that a query belongs to a particular gesture class before it is rejected—we discard queries when their measure is above the threshold, and accept them when their measure is below,

contingent on the satisfaction of any additional application-dependent criteria. Usual techniques that practitioners use to decide an appropriate threshold are impractical under typical customization conditions, given that so little data is available from which the system can learn. To illustrate with linear discriminate analysis, one can use the Mahalanobis distance to reject queries measuring three standard deviations away from the class mean [215], but to reliably estimate the covariance matrix requires more training data than is often available. In our previous work [235], we overcame this issue by estimating rejection thresholds from synthetically generated score distributions for low-activity data.

In other words, with sufficient training data, a recognizer can estimate within class score probability densities and select thresholds that improve recall. One can also use negative (non-gesture) sequences to select thresholds that improve precision. With access to both positive and negative samples, one can select thresholds that strike a balance between recall and precision. However, as is the nature of custom interface design, we must assume that only a few positive (but not negative) training samples are given per gesture class. By using synthetic data generation, we can generate positive and negative sample distributions from which we may derive reasonable threshold values.

In this chapter, we complete The Dollar General continuous dynamic custom gesture recognition pipeline by introducing an automatic rejection threshold mechanism we call The Voight-Kampff Machine. VKM comprises two components that build on our previous work: GPSR (Chatper 8) and Mincer. We specifically develop a new GPSR optimal-N resampling rate selection function that allows us to synthesize within class score distributions; and Mincer improves on splicing (Appendix A) to enable class-specific negative sample synthesis. VKM uses both GPSR and Mincer to generate score distributions from which a rejection threshold is selected.

9.1 Voight-Kampff Machine: Rejection Threshold Selection

As outlined above, our rejection threshold selection strategy is to generate positive and negative synthetic score distributions from training data. Using these distributions, we can estimate accuracy over varying threshold scores and select that which maximizes accuracy. In prior work (see Appendix A), we described one such approach that, in retrospect, suffers from two issues. First, we generated per template, z-score normalized distributions that we combined into common negative and positive distributions. We then located the common z-score value that maximized separation between these distributions, and converted the value into a per-class rejection threshold using per-template mean and variance statistics. This approach has potential to cause bias, where non-uniform gesture class vocabularies can result in non-optimal thresholds. For example, if there is one kick gesture in a set of mostly arm and hand gestures, the kick gesture's negative score distribution is likely to be further from zero than will be the upper body gestures, resulting in the kick gesture having a lower rejection threshold. A second issue is that our prior accuracy measure did not have an intuitive interpretation given our z-score transformation strategy.

Moving forward, we address both issues but continue to use a similar process, whereby we select rejection thresholds from synthetically generated score distributions. One major difference is that we no longer z-score normalize results. Instead, untransformed scores are combined directly into their common score distributions. This decision is motivated by the success of prior work that uses common covariance matrices in linear discriminant analysis [22, 40, 215, 236]—we similarly assume the distribution of samples surrounding class means share a common shape across classes. With this approach, one can directly interpret results and the generation process is simplified. To address the second issue, we use a standardized accuracy measure called F_β-score that we discuss shortly. The overall process is summarized in Algorithm B.19.

9.1.1 F_β-Score

To select a rejection threshold, we estimate F_β-score from synthetic data. F_β-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_β-score also has the desirable property that it does not include true negative results in its calculation. This property is important because continuous input contains mostly negative data, and optimizing a system for high true negative detection may result in poor threshold selection. Formally, $F_\beta-$score is defined as follows:

$$F_\beta = \left(1+\beta^2\right) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{9.1}$$

$$= \frac{(\beta^2+1)\, tp}{(\beta^2+1)\, tp + \beta^2 fn + fp}, \tag{9.2}$$

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:

$$\text{precision} = \frac{tp}{tp+fp}, \tag{9.3}$$

and

$$\text{recall} = \frac{tp}{tp+fn}, \tag{9.4}$$

and β controls the relative weight between precision and recall. When $\beta = 1$, we have the traditional F_1-score that equally balances both. 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.

As shown in Algorithm B.20, it is trivial to estimate F_β from two lists, one storing positive scores

and the other negative. Specially, we first sort the lists and then iteratively estimate F_β by tracking classification results as we simultaneously walk both lists. The set of positive scores that fall below a certain threshold are true positives, whereas the remaining are false negatives. Similarly, those negative scores that fall below the same threshold are false positives, whereas those above are true negatives. After examining all possible thresholds, we simply return the threshold associated with the highest F_β-score.

9.1.2 Rejection Threshold Scaling

We now address two issues that impact rejection threshold selection. First, as we discuss in Chapter 12, the data collection protocol one uses to collect training data impacts gesture production variability. A common data collection approach practitioners use is to collect isolated samples that users generate in a low-stress environment, one at a time. The variability of such data is likely to be less than that of the target application, where differences in environment, stress, usage, and general sloppiness collectively contribute to greater variability. Since our technique attempts to locate gesture class boundaries from given training data, we propose to inflate the threshold by a value determined a priori. The **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 9.1, thresholds must be sufficiently large so as to provide adequate coverage over the application gesture class space. However, 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.

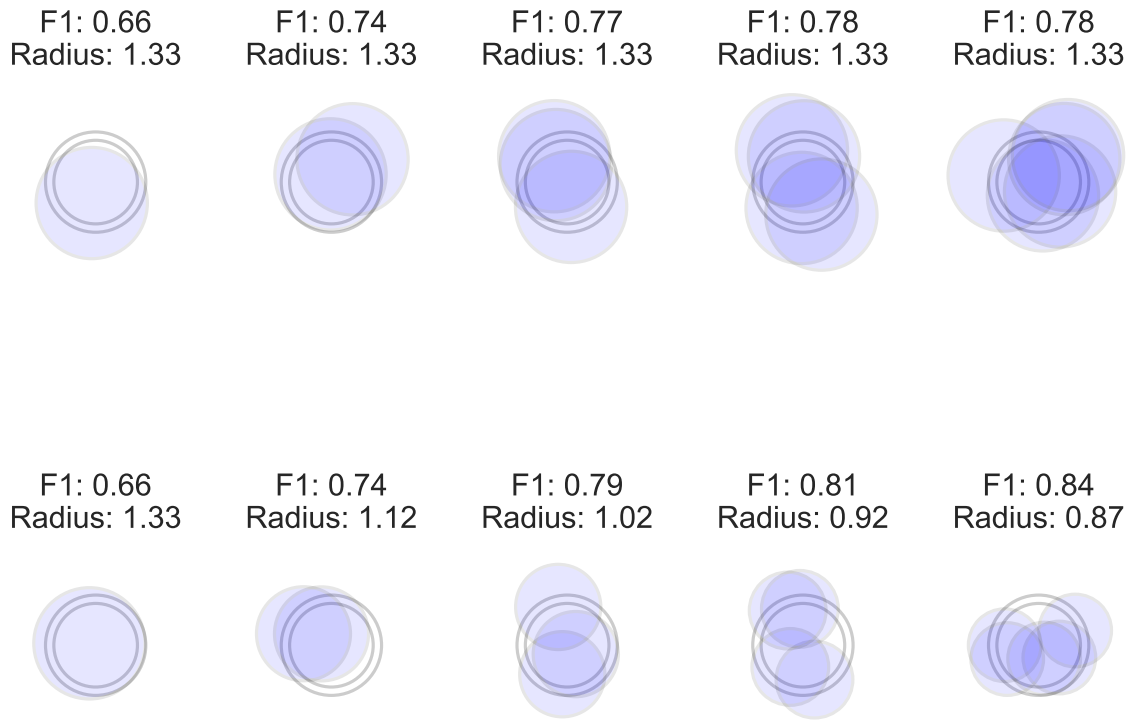


Figure 9.1: Simulated effect of threshold scaling on F_1-score. Each subfigure comprises two rings that enclose the training and application gesture class spaces, refered to as the inner and outer rings, respectively. Individual training samples are randomly drawn from the inner space and rendered as 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.

9.1.2.1 Simulation-based Rejection Threshold Adjustment

We use simulation to estimate the rejection threshold scale; see Algorithm B.21. Our simulation accepts an initial threshold τ, an inflation factor ι, and the training set size T; and the simulation outputs an adjusted threshold. Specifically, our goal is to find a rejection threshold scaling factor λ that maximizes F_β. We represent the gesture class training sample space as a hypersphere centered

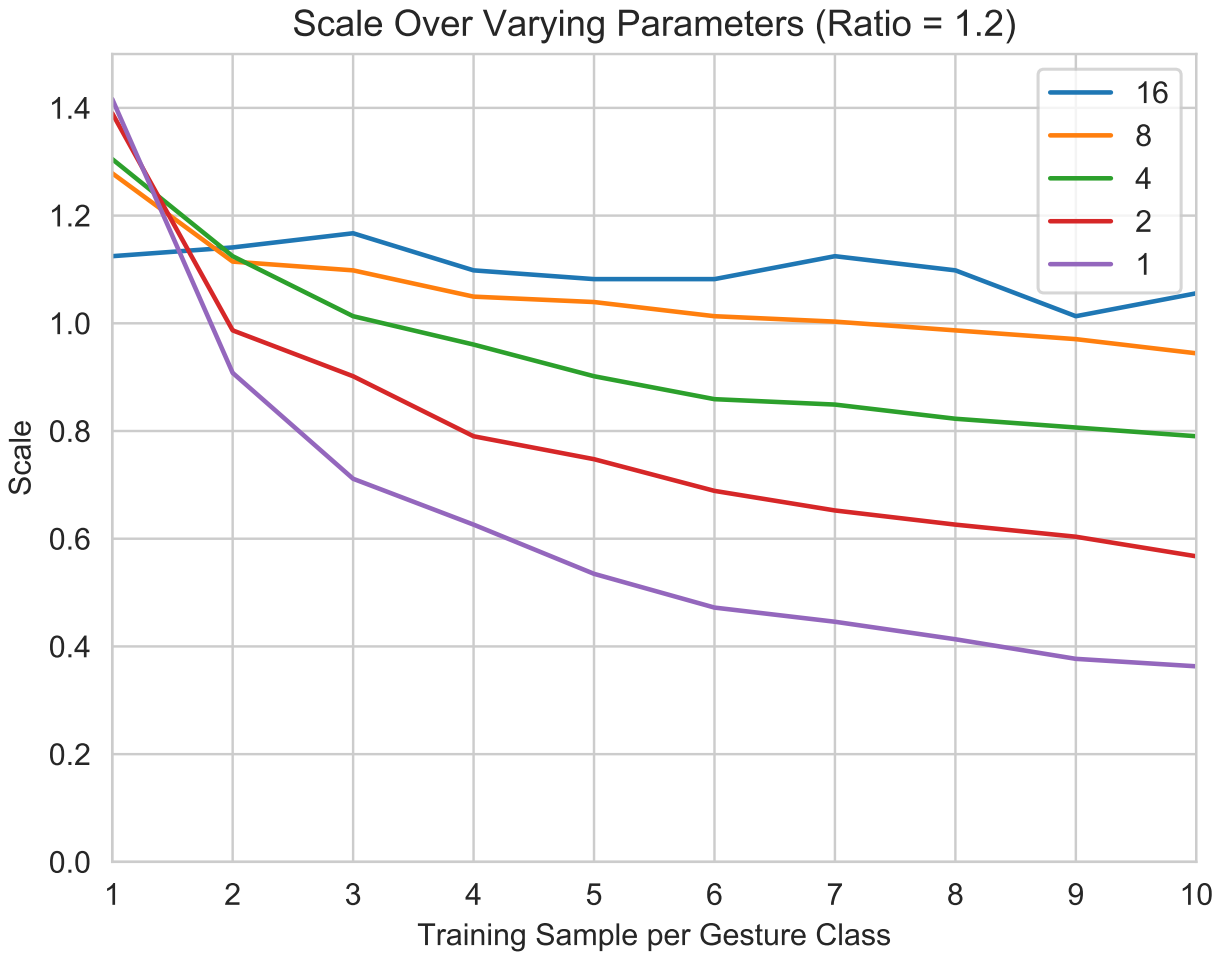


Figure 9.2: Reduction in scale as training set size increases over varying dimensions. Notice that the difference in scale declines as the number of dimensions increase.

on zero whose radius is the given rejection threshold τ. 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 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 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 as expected, a false positive point is one that is recognized but falls outside of the application space. To estimate an F_β-score, we generate a large number of test cases, tally classification results, and estimate accuracy from the results. To select an adjusted threshold, we iterate over a number of scaling factors λ and select that which maximizes accuracy.

One remaining problem is that we must select the hypersphere's dimensionality. Suppose we spatially resample 63-component Kinect trajectories to 16 points. We then have a 1008-dimensional point. Sampling points in this space is both computationally prohibitive and unrepresentative of gestures because it assumes complete independence between components. In reality, data within trajectories are highly correlated—we need only a few points to identify a gesture's class [245]. For example, Vatavu found that 2D gestures resampled to six points yield almost optimal recognition performance under Euclidean distance [245], and similiar results were reported for 3D gestures [247]. We also observe via Figure 9.2 that the effect of dimensionality on scaling diminishes logarithmically as dimensionality increases. For these reasons we use $d = 6$ in our evaluations, though we expect higher values will also work well.

One final note is that although we provide an automatic rejection threshold adjustment technique, we are able to provide common values one can use in a look up table, and so practitioners should not have to implement our simulation. We have now defined a general framework for rejection threshold selection. Next, we turn our attention to negative sample synthesis first and positive sample synthesis, thereafter.

9.2 Voight-Kampff: Mincer

We describe two synthetic negative sample data generation techniques in this section, one that generalizes our earlier splicing technique [235], and a second called Mincer. We found that although splicing adequately reproduces negative score distribution within our framework, it does not lead to suitable rejection thresholds for high-activity data. We first describe splicing and its shortcomings.

9.2.1 Splicing

To generate negative data, one may "splice" together positive training samples to create semi-nonsense, noise-like sequences. We favor this approach because we desire negative samples that have real gesture data embedded within them, ensuring our system learns to reject those sequences that partially resemble real gestures. In prior work, we simply stitched together randomly sampled subsequences from training data. However, we now describe this process in formal terms. We also generalize splicing to describe a stream of negative data as follows:

$$Y = X_1 X_2 X_3 \ldots, \tag{9.5}$$

where each X_i $(i \in \mathbb{N})$ is a subsequence:

$$X_i = H(T_j[\min(a,b) : \max(a,b)]) \tag{9.6}$$

$$j = \mathscr{U}(1, |\mathbb{T}|) \tag{9.7}$$

$$a = \mathscr{U}\left(1, \left|T_j\right|\right) \tag{9.8}$$

$$b = \mathscr{U}\left(1, \left|T_j\right|\right). \tag{9.9}$$

$\mathscr{U}$ denotes a sample drawn from the discrete uniform random distribution, and H is a perturbation function that accepts and returns a time series. In words, X_i is a subsequence of the randomly selected training sample T_j that has been (possibly) modified by H. We intend Y to be treated like an input device stream so that an application can draw samples online, in realtime, as needed.

Now, let $\mathbb{H}$ be a set of transformation functions defined as:

$$\mathbb{H} = \{h_i \mid 1 \leq i \leq \mathbb{H}_n\} \tag{9.10}$$

$$h_i : \mathbb{R}^{N_1 \times m} \rightarrow \mathbb{R}^{N_2 \times m}, \tag{9.11}$$

where each function h_i accepts as its single parameter a time series of length $N_1 > 0$ and outputs a modified variant of the input having length $N_2 > 0$. Note, there is no restriction on the relationship between N_1 and N_2, as the series may grow or shrink during a transformation. With $\mathbb{H}$ in hand, we can define perturbation function H recursively as follows:

$$H_{i-1}(X) = \begin{cases} X & \text{if } u_i = 0 \\ h_{u_i}(H_i(X)) & \text{otherwise} \end{cases} \tag{9.12}$$

where $U = \{u_1, u2, \ldots\}$ is a random sample drawn from the population $\mathscr{U}(0, \mathbb{H}_n)$.

9.2.1.1 A Closer Look At Splicing

To understand splicer's negative sample generation potential, we compare its output to that of the high-activity datasets described in Chapter 7. Specifically, for a given participant, we train Jackknife with one random training sample from each gesture class. One hundred times, thereafter, we randomly sample a subsequence from the participant's session data and measure its similarity to all templates. This entire process is also repeated one hundred times per participant, and all scores

are combined into the *real* negative sample score distribution. Note, we acknowledge there is small, but non-zero probability that a random session sample will precisely contain a valid gesture. We treat this circumstance as a low probability event that results in inconsequential noise one can safely ignore.

Using the same procedure just described, we generate a synthetic data distribution via splicing. This forms our synthetic score distributions. Figure 9.3 reveals our findings. We observe that the real and synthetic distributions are remarkably close. Using the Kolmogorov–Smirnov statistic to measure the statistical distance between distributions, we find that $D = 0.07$ in the worst case (Mouse data) and $D = 0.035$ in the best (Kinect data). In other words, the maximum divergence in the cumulative distributions between real and synthetic scores is 7% and 3.5%, respectively. Overall, we find this divergence acceptable in the context of estimating rejection thresholds, especially since maximum divergence occurs outside of the positive sample distribution range.

Note our splicing solution was implemented with support for reversing sequences. We were prepared to investigate other perturbation functions based on affine transformations, Fourier coefficient manipulations, and GPSR, among others. However, because the statistical distance between distributions were found to be already adequate, it was unnecessary to further pursue.

Results in Appendix A indicate that splicing works well for low-activity data, and an analysis of negative score distributions reveal why this may be the case. Non-gesture activities form similarity scores that have low tail densities and substantial mass near each domain's center. Splicing is able to replicate these properties with sufficient accuracy. Since positive sample scores cluster near the lower tail, and because we previously used window-based segmentation with ad-hoc heuristics, one can select rejection thresholds between distributions.

We believe that our prior ad-hoc criteria is key in understanding why we were able to use splicing-based rejection threshold selection. Our criteria required several consecutive positive results, and

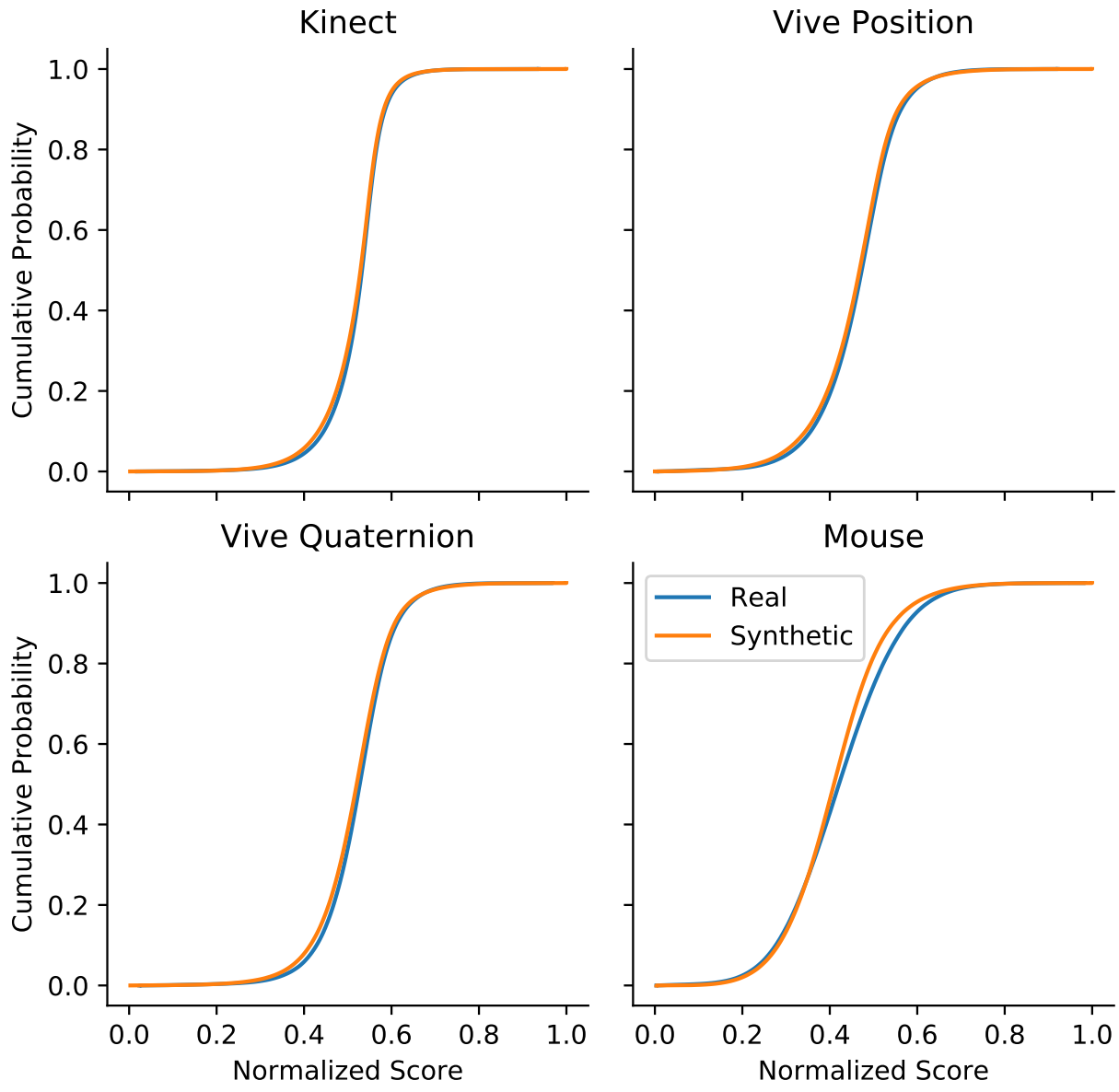


Figure 9.3: Comparison of real and splice-based negative score distributions over four high-acitivity datasets. Measurements were made between training and negative samples using Jackknife. Although the real and synthtyic distributions are statistically different from each other according to two-sample Kolmogorov–Smirnov testing, we see they are sufficiently close for our use.

once a gesture was classified, we disabled recognition for a short period thereafter. With a sufficiently large window and idle motion between gestures, we may assume a gesture will appear within several consecutive windows and each window will yield a positive result. Noise and subtle motions on either side of a gesture will increases the score, but because our rejection threshold lies somewhere between the idle and positive sample score distributions, it still falls below the rejection threshold.

However, as we were able to demonstrate in Chapter 7, we cannot use a single large window with high-activity data. This requires an adjustment to our threshold that results in substantially inferior performance. Since we do not have this problem with low-activity data, we believe this is why we were able to achieve high recognition accuracy with thresholds selected using splice-based synthesis. In the next section, we propose Mincer as a way to overcome this problem.

9.2.2 Mincer

One may think of Mincer as being a specialized type of splicing. As just discussed, we found that the distance between positive and negative distributions were too far apart to allow for reliable rejection threshold estimation in the presence of high-activity data. To correct this problem, we need to generate negative samples that are closer to a given training sample—negative samples that fall near the class boundary. To accomplish this, we propose to replace subsets of a given trajectory with training data sampled from different gesture classes.

In more detail, we spatially resample all training data to high resolution trajectories, ensuring we preserve local features, and min-max normalize each sample. Thereafter, we convert each trajectory to a set of direction vectors, storing motion rather than position patterns. To generate a negative sample via mincing, we first randomly select a prepared sample from a different class and two indices. We then copy the associated subset of direction vectors from the selected sample

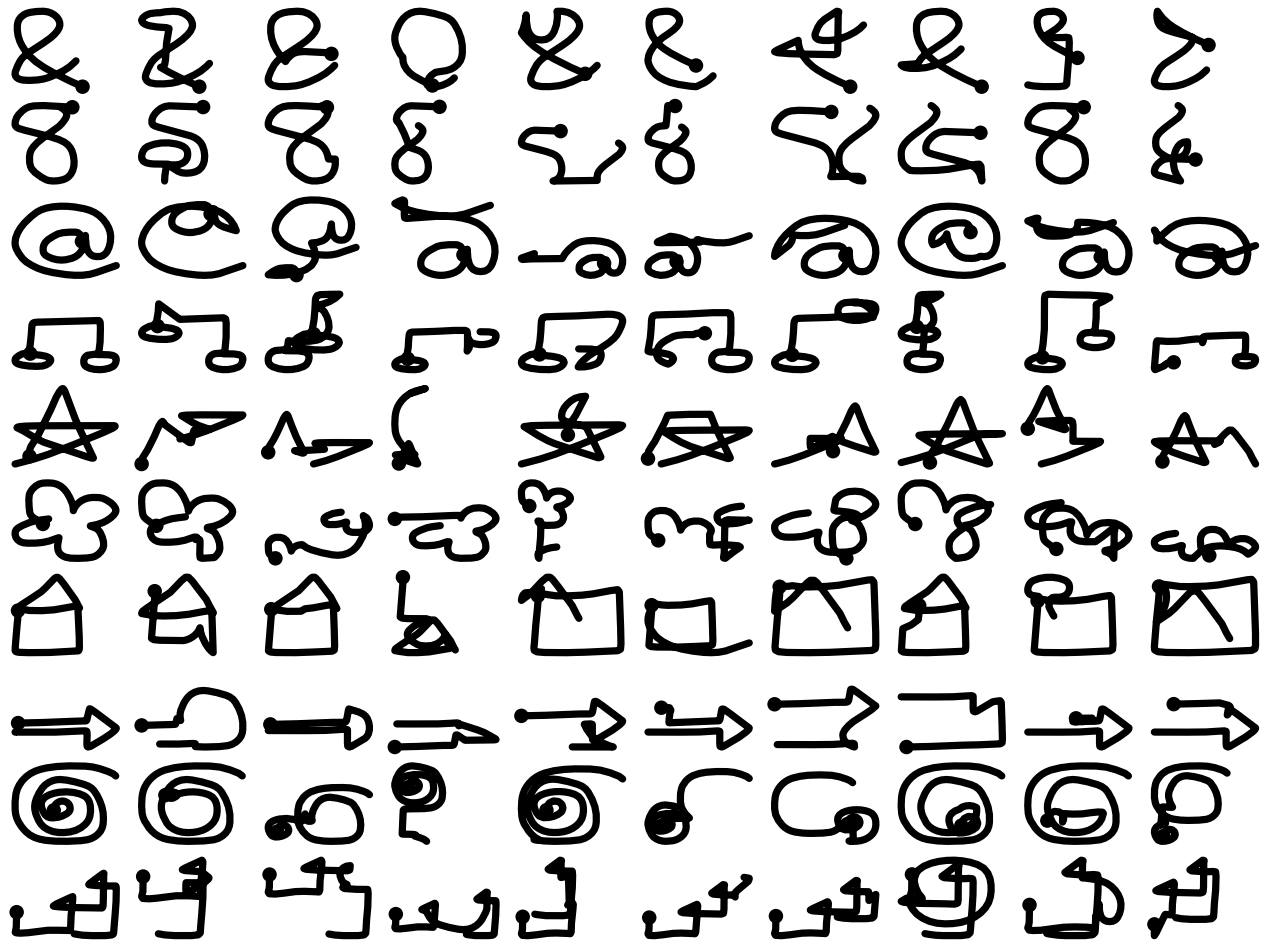


Figure 9.4: Example negative samples generated with Mincer. The first column comprises training samples from the high-activity Mouse datasest followed by subsequent minced variants. Notice how significant portions of the original seed samples appear in their negative sample counterparts. This similarity allows us to find the seed sample's class boundary when measured by a recognizer.

into the target sample at the same location specified by the indices. When the difference between indices is negative, we copy direction vectors in reverse order. Last, we perform a cumulative summation operation to convert direction vectors into points. Mincer pseudocode is listed in Algorithm B.23, and example mincings are illustrated in Figure 9.4.

Initially, we believed it would suffice to uniformly sample indices from across the full range of allowable values. We found, however, that width matters. If the resulting distribution contains too many minced samples with negligible modification 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 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.

9.3 Voight-Kampff: Gesture Path Stochastic Resampling

In this section we describe how we use gesture path stochastic resampling (GPSR) to generate representative positive score distributions. Since we use the KS statistical distance measure to optimize GPSR equation coefficients, we first introduce the KS statistic.

9.3.1 The Kolmogorov–Smirnov (KS) Statistical Distance Measure

The Kolmogorov–Smirnov (KS) test is a popular nonparametric test that one can use to evaluate whether two random samples are drawn from the same distribution. At its core lies the KS statistic $D \in [0,1]$, which measures the statistical distance between two distributions. Low values of D indicate that the random samples under investigation are drawn similarly if not identically. Since D is easy to compute and does not assume an underlying structure, we will use this statistic to measure the difference between real and synthetic score distributions.

In more detail, the empirical distribution function of a random sample X comprising n elements is defined as:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{[-\infty,\, x]}(x_i), \tag{9.13}$$

where I is the usual indicator function:

$$I_{[a,b]}(x) = \begin{cases} 1 & a \leq x \leq b \\ 0 & \text{otherwise.} \end{cases} \tag{9.14}$$

The KS statistic $D_{n,m}$ over two random samples X_1 and X_2 with empirical distributions $F_{1,n}$ and $F_{2,m}$ having lengths n and m respectively is defined as:

$$D_{n,m} = \sup_{x \in \mathbb{R}} |F_{1,n}(x) - F_{2,m}(x)|. \tag{9.15}$$

In words, $D_{n,m}$ measures the maximum distance between two cumulative distributions. Our objective is therefore to minimize $D_{n,m}$ by manipulating GPSR parameters so as to bring the synthetic and true distributions closer together.

9.3.2 Gesture Path Stochastic Resampling, Revisited

In short review of GPSR, we first spatially resample a given trajectory so that the distance between selected points along the gesture path is random, rather than uniform. We then rescale 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. For more information, see Chapter 8.

In VKM, we use GPSR 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. Since 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.

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. We previously found an optimal-N solution for 2D gesture data based on density and openness, but we found this to be inadequate for other input device types. We were also concerned with realism rather than score distributions. For these reasons, we require a new equation.

9.3.2.1 Optimal-N for Synthetic Score Distributions

We sought to find an optimal resampling rate based on trajectory properties such that synthesized samples reproduce the within class variance when measured against its seed trajectory. 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 [22, 215, 232], 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 (See Chapter 7) 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.

We now describe our optimization procedure. First, for each high-activity dataset training sample, 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 there are 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 distributions matches the target's variability. That is, for each N, we generate four synthetic samples using N and measure their distances from the seed sample; these scores are then combined into variability results we compare against the target value. 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 follow equation:

$$n = x_0 + x_1 \cdot \text{density} + x_2 \cdot \text{ angle } + x_3 \cdot \text{density} \cdot \text{angle} \tag{9.16}$$

We used Luus–Jaakola (LJ) optimization [151] 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, afterwhich we switch to the Kolmogorov–Smirnov distance.

This process produced a unique set of coefficients per input device type. The resulting cumulative probability distributions are shown in Figure 9.5. 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. In Figure 9.6, we plot the within class score of every training sample from every dataset. Visual inspection 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 (forth column) scores are significantly larger than real for 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 9.5), we believe the current equation meets our requirements. We next verify this assumption through an extensive evaluation of VKM.

9.4 Evaluation

In this section, we evaluate The Dollar General's efficacy on high-activity data. We are specifically interested in The Voight-Kampff Machine's ability to select rejection thresholds relative to other techniques. To this end, we discuss our dataset, alternative techniques, testing methodology, and then we present our findings.

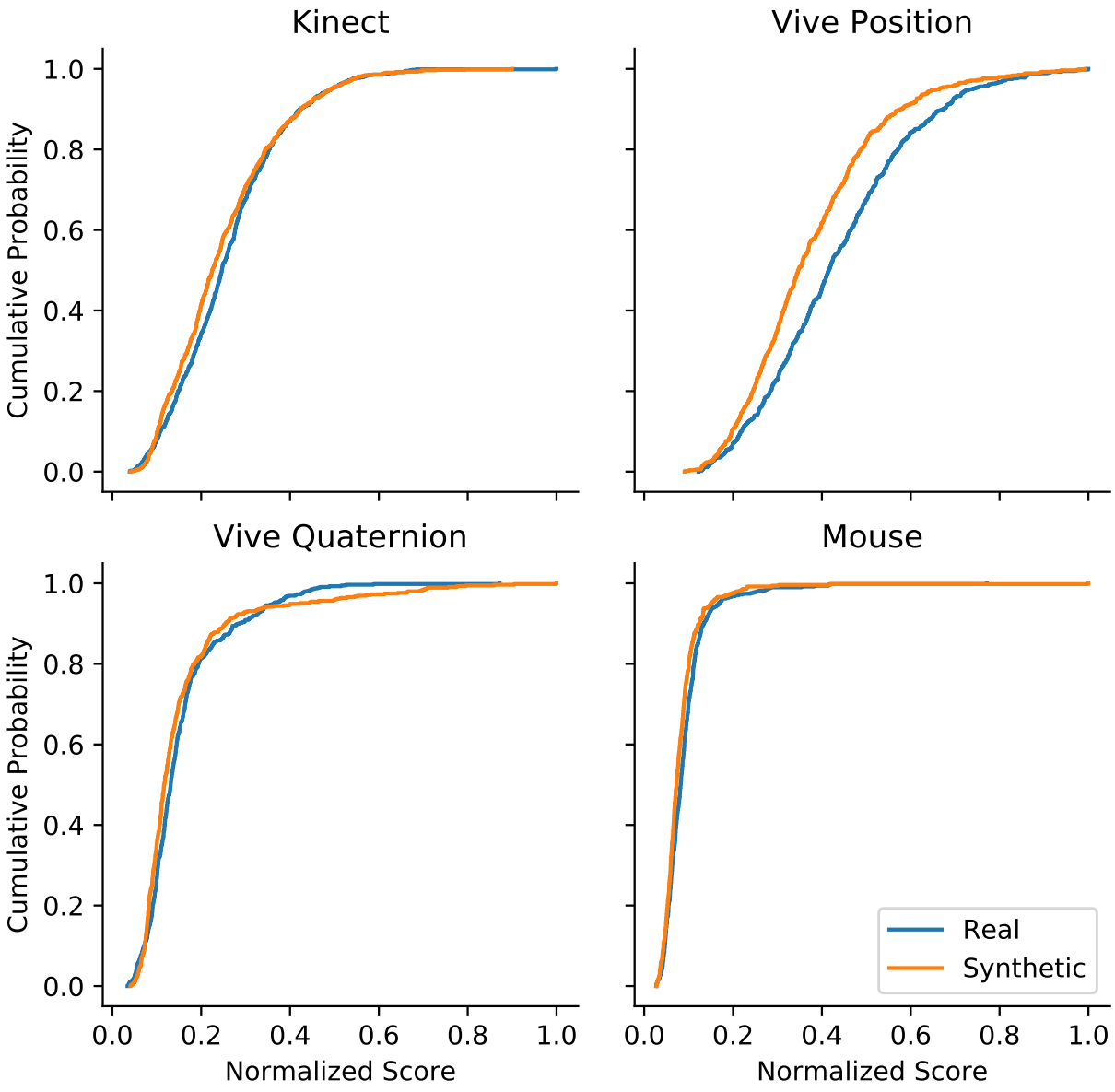


Figure 9.5: Cumulative probability distribution of the within class scores measured by Jackknife for real and GPSR synthesized gestures.

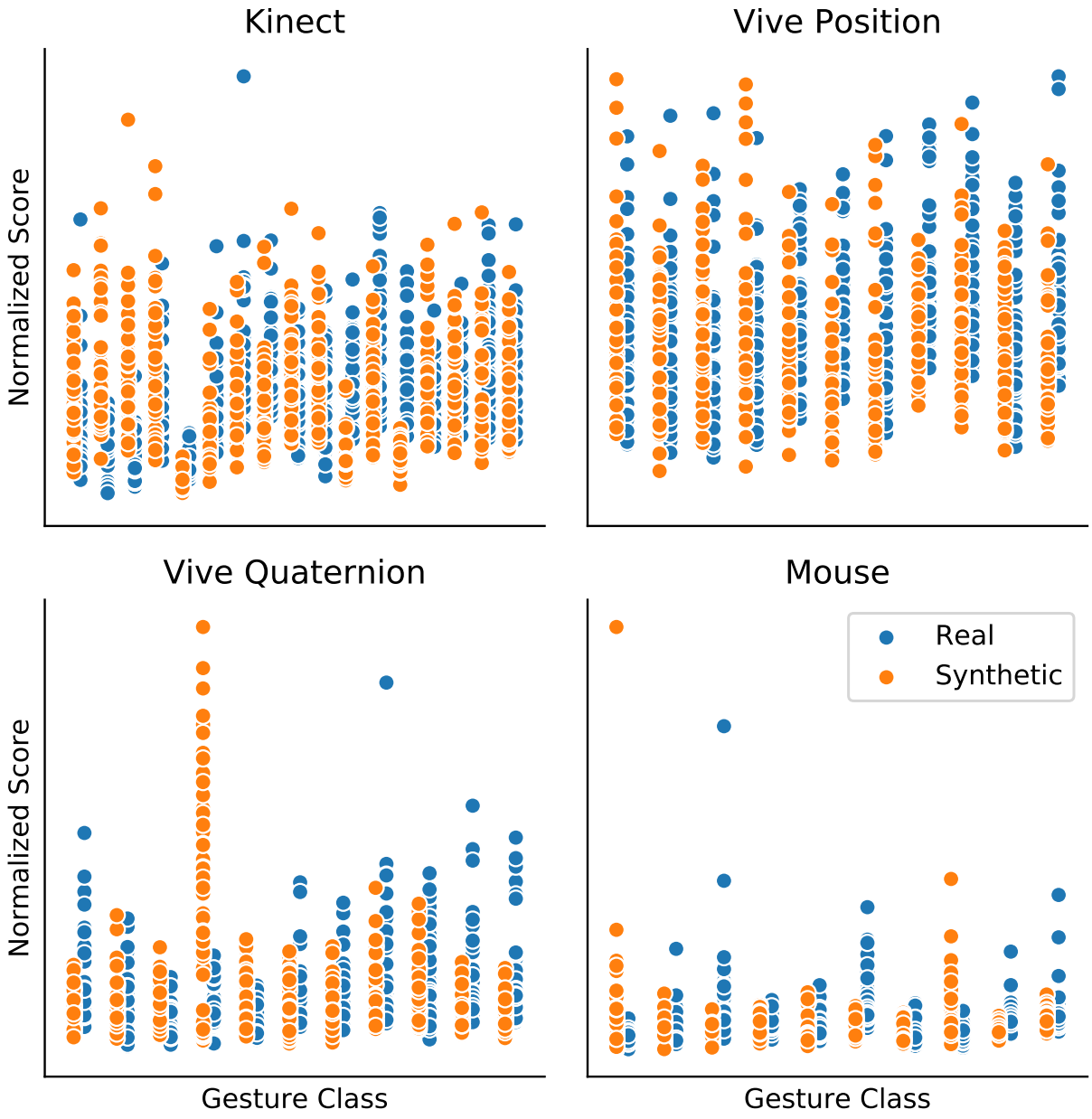


Figure 9.6: Mean within class score of each training sample to every other sample (blue) and of each training sample to an equal number of synthetically derived samples (orange). Gesture classes are enumerated and organized along the x-axis so that real and synthetic distributions are collocated by gesture.

9.4.1 High Activity Data

To compare rejection threshold selection strategies, we use the high-activity (HA) dataset described in Chapter 7. To recap, we collected five training samples from ten participants per input device type: Mouse, Vive, and Kinect. (Vive data is further subdivided into position and orientation datasets.) We thereafter collected continuous data from each participant using a Follow-The-Leader (FTL) protocol [234], whereby participants replicate the motions of a virtual actor, *i.e.*, leader, who continuously performs random actions. Our data collection tool periodically interjects gesture requests via an on screen text prompt that the participant must immediately observe and execute, regardless of the actor's state. After gesticulating, the participant must, without pause, return to following the leader. In the end, we collected three examples of each gesture class interleaved with actions that resemble puppeteering, direct object manipulations, and other non-gesture actions. Our penultimate goal is to recognize all embedded gestures with no false positives.

9.4.2 Rejection Threshold Selection Techniques

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

1. **Iterative Search**: With enough training data, one can trivially iterate through the rejection threshold parameter space and select that which maximizes accuracy across all participants per training set size T. This selection technique does not cross validate nor is it appropriate for customization. We include iterative search so that it may serve as a baseline for near maximum performance (and for this reason, we hereafter refer to iterative search as the baseline).

2. **Mincer**: The technique described in this chapter whereby we use GPSR to construct a pos-

itive distribution and Mincer to construct a negative distribution, and from which we select the threshold that maximizes accuracy.

3. 3σ: Random samples are often assumed to follow a normal distribution and that results within three standard deviations (3σ) of their mean include 99.7% of all samples. Even when distributions are not normal, practitioners sometimes use a 3σ pruning rule. For this reason, we include one method that uses GPSR to estimate the positive distribution mean and standard deviation, and then set the rejection threshold to be 3σ above the mean.

4. 3σ-per-class: This technique is identical to 3σ, except estimates and rejection thresholds are per class, rather than global.

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 [267] 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 the square, curly, and rounded brackets, 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 Mincer, except that we use generalized splicing to generate the negative score distribution.

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.

9.4.3 Test Procedure

We engage in user dependent recognition testing similar to that discussed in prior chapters. 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 Dollar General and record all recognition results. From hand-labeled ground truth data, we analyze classification results to generate an F_β score. This process is repeated ten times per participant and all results are averaged into a single accuracy measure.

9.4.4 Analysis

With only 10 participants per input device, power to detect differences between multiple conditions and levels is limited. For this reason, we decided in advance to conduct a single 1-factor repeated measures experiment per input device type, where threshold selection technique was the factor. However, we limit our statistical analysis to only the top four performers (Baseline, Mincer, 3σ, 3σ-per-class), although we report all results. Our dependent variable was F_1-score. To determine if the selection technique is significant for a given input device, we used the Friedman test procedure [77]. Afterward, we used exact, two-tailed Wilcoxon signed rank tests to detect pairwise differences between selection methods. To protect against multiple comparison type I errors, we used the Holm-Bonferroni step-down procedure [100].

For each input device type, we render a line graph that shows all selection techniques over the varying training set sizes $T \in [1,5]$ with 68% confidence bands, to help with legibility. Exact means and standard deviations accompany each graph in a subsequent table. In a second table, we

report the percentage increase in error rates from baseline to each selection method. Finally, we report post-hoc results in a third table per input device.

9.4.5 Results

9.4.5.1 Kinect

Kinect accuracy results are shown in Figure 9.7 and Table 9.1. Baseline, 3σ, 3σ-per-class, and Mincer are all statistically different from each other (Table 9.3). We find that Baseline and Mincer achieve high accuracy ($\geq 90\%$) with a single training sample per gesture class and they both improve as the training set size increases; although performance appears to plateau at $T = 3$. The difference in errors rates between these two methods is low (Table 9.2). Note that differences in error rate at high accuracies will have dramatic impacts on percentage error increases, *e.g.*, Mincer obtained a 9% increase in its error rate at $T = 5$, yet Baseline and Mincer both achieve approximately 95% accuracy. The remaining techniques do not perform as well, though 3σ-per-class straddles the high accuracy threshold.

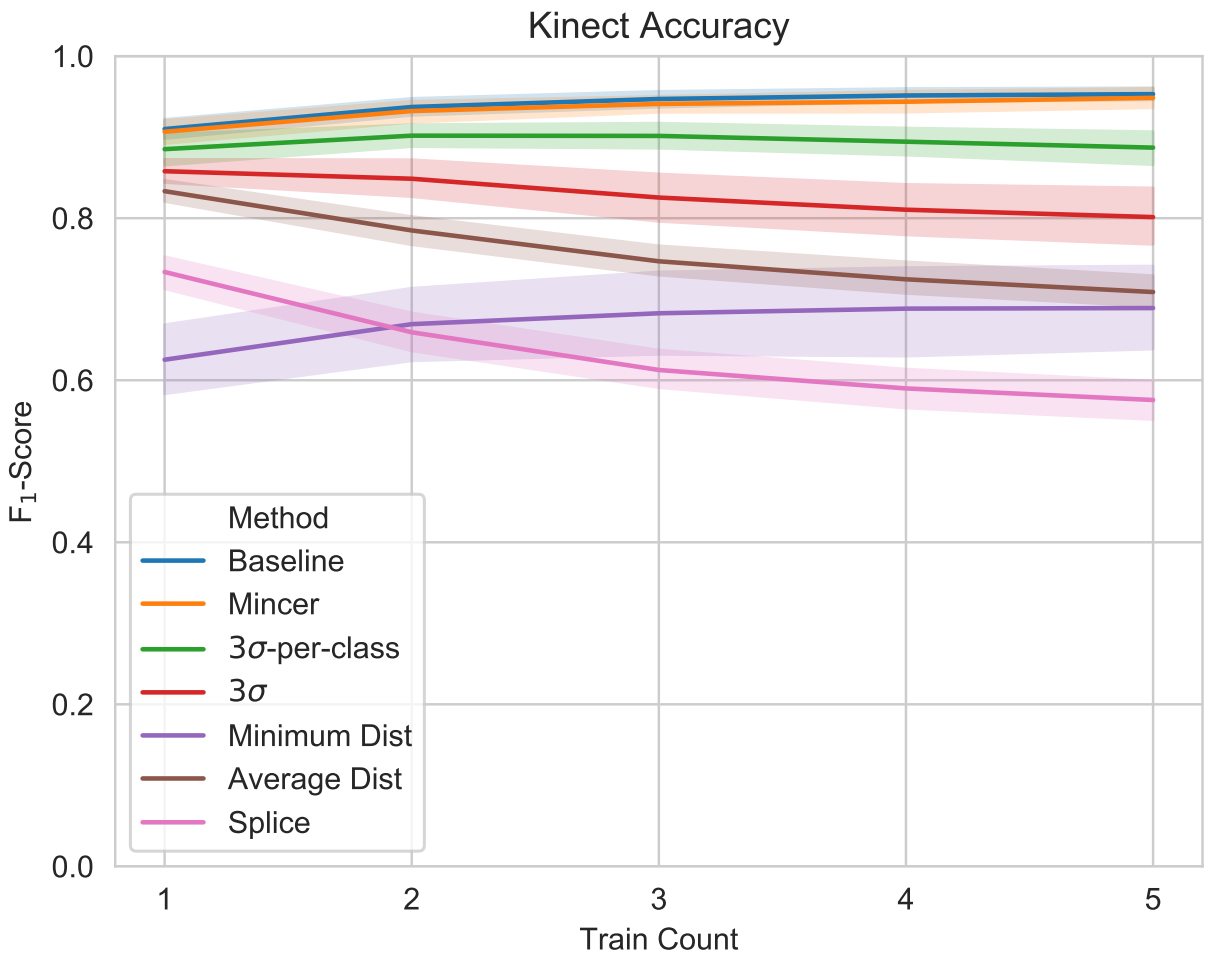


Figure 9.7: F_1-score over varying methods and training set sizes for Kinect. Exact values are given in Table 9.1 below. Confidence bands are standard error (68%) for legibility.

Table 9.1: F_1-scores over varying methods and training set sizes for Kinect.

Method	T=1		T=2		T=3		T=4		T=5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Baseline	0.91	0.04	0.94	0.03	0.95	0.03	0.95	0.03	0.95	0.02
Mincer	0.91	0.05	0.93	0.04	0.94	0.03	0.94	0.04	0.95	0.04
3σ-per-class	0.89	0.06	0.90	0.04	0.90	0.05	0.89	0.05	0.89	0.06
3σ	0.86	0.05	0.85	0.07	0.83	0.09	0.81	0.10	0.80	0.11
Minimum Dist	0.63	0.13	0.67	0.15	0.68	0.16	0.69	0.16	0.69	0.16
Splice	0.73	0.06	0.66	0.07	0.61	0.07	0.59	0.07	0.58	0.08
Average Dist	0.83	0.04	0.78	0.05	0.75	0.06	0.72	0.06	0.71	0.06

Table 9.2: Percentage increase in error rates over baseline for Kinect.

Method	T=1	T=2	T=3	T=4	T=5
Mincer	3	7	11	15	9
3σ-per-class	27	56	86	117	140
3σ	57	141	230	290	323
Minimum Dist	316	427	500	541	563
Splice	196	443	632	744	805
Average Dist	85	242	378	467	521

Table 9.3: Kinect accuracy post-hoc results using the exact, two-tailed Wilcoxon signed rank test for top four results. Note, Friedman testing revealed a significant difference between methods, $\left(\chi_3^2 = 22.3, p < .001\right)$.

Pair	$W+$	p	r
Baseline - 3σ	0	$< .05$	1.0 L
Baseline - 3σ-per-class	0	$< .05$	1.0 L
Baseline - Mincer	0	$< .05$	1.0 L
3σ - 3σ-per-class	50	$< .05$	0.8 L
3σ - Mincer	52	$< .05$	0.9 L
3σ-per-class - Mincer	51	$< .05$	0.9 L

9.4.5.2 Vive Position

Vive Position accuracy results are shown in Figure 9.8 and Table 9.4. As before, Baseline, 3σ, 3σ-per-class, and Mincer are all statistically different from each other (Table 9.6). However, unlike before, we find that all methods except Average Dist and Splice achieve high accuracy ($\geq 90\%$) with a single training sample per gesture class, and improve as the training set size increases. Though, performance appears to plateau at $T = 3$. Based on percentage error rate increases (Table 9.5), differences between the top performers are minor expect at $T = 1$, where Mincer is notably closer to Baseline performance relative to the other methods.

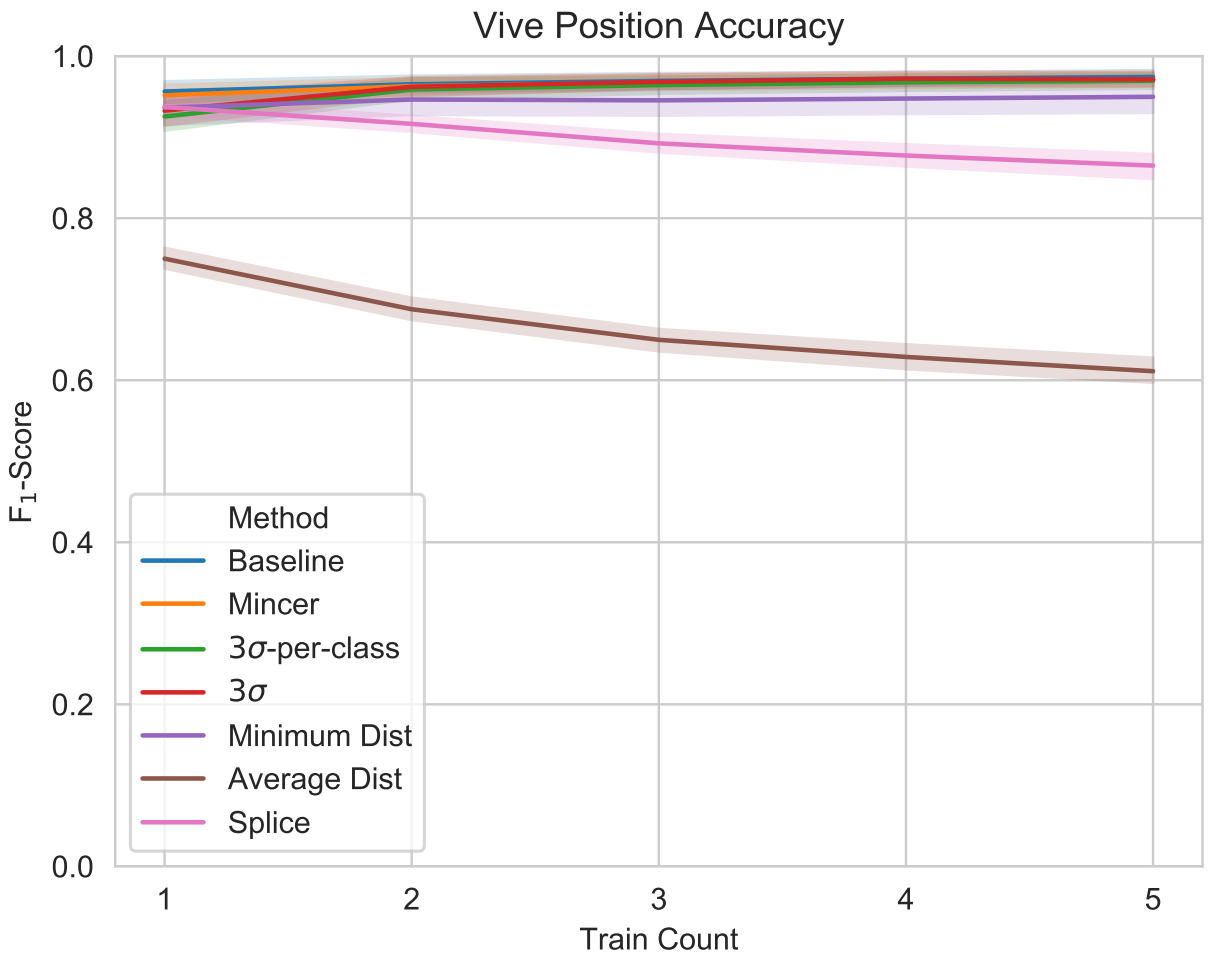


Figure 9.8: F_1-score over varying methods and training set sizes for Vive Position. Exact values are given in Table 9.4 below. Confidence bands are standard error (68%) for legibility.

Table 9.4: F_1-scores over varying methods and training set sizes for Vive Position.

Method	T=1		T=2		T=3		T=4		T=5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Baseline	0.96	0.04	0.97	0.03	0.97	0.03	0.97	0.03	0.97	0.02
Mincer	0.95	0.04	0.96	0.03	0.97	0.03	0.97	0.03	0.97	0.03
3σ-per-class	0.93	0.06	0.96	0.04	0.96	0.03	0.97	0.03	0.97	0.03
3σ	0.93	0.06	0.96	0.03	0.97	0.03	0.97	0.03	0.97	0.03
Minimum Dist	0.94	0.06	0.95	0.06	0.95	0.06	0.95	0.06	0.95	0.06
Splice	0.94	0.03	0.92	0.03	0.89	0.04	0.88	0.04	0.86	0.05
Average Dist	0.75	0.04	0.69	0.04	0.65	0.04	0.63	0.05	0.61	0.05

Table 9.5: Percentage increase in error rates over baseline for Vive Position.

Method	T=1	T=2	T=3	T=4	T=5
Mincer	10	6	7	8	12
3σ-per-class	70	18	16	16	17
3σ	54	8	3	0	11
Minimum Dist	45	54	78	88	95
Splice	43	140	252	342	426
Average Dist	472	799	1045	1238	1416

Table 9.6: Vive Position accuracy post-hoc results using the exact, two-tailed Wilcoxon signed rank test for top four results. Note, Friedman testing revealed a significant difference between methods, $\left(\chi_3^2 = 19.6, p < .001\right)$.

Pair	$W+$	p	r
Baseline - 3σ	0	$< .05$	1.0 L
Baseline - 3σ-per-class	0	$< .05$	1.0 L
Baseline - Mincer	0	$< .05$	1.0 L
3σ - 3σ-per-class	50	$< .05$	0.8 L
3σ - Mincer	52	$< .05$	0.9 L
3σ-per-class - Mincer	51	$< .05$	0.9 L

9.4.5.3 Vive Quaternion

Vive Quaternion accuracy results are shown in Figure 9.9 and Table 9.7. As before, Baseline, 3σ, 3σ-per-class, and Mincer are all statistically different from each other (Table 9.9). Though, with this dataset, only Mincer and 3σ is able to achieve high accuracy, and only at $T = 5$. The second best overall performer, 3σ-per-class, is on par with Mincer when the training count reaches $T = 4$, but under performs before this point. For both techniques, accuracy continues to improve as the training count grows, and given more samples, 3σ may over take Mincer.

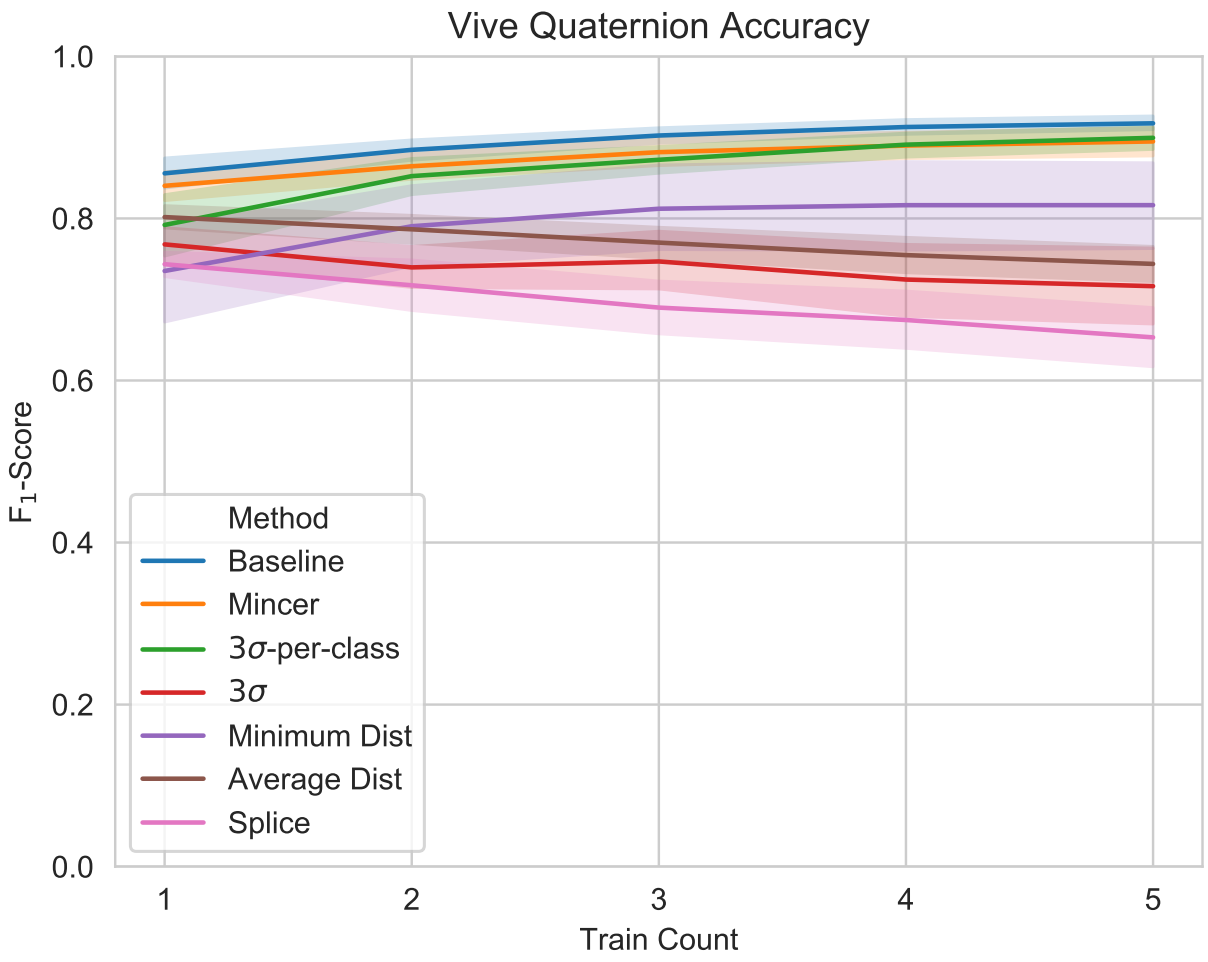


Figure 9.9: F_1-score over varying methods and training set sizes for Vive Quaternion. Exact values are given in Table 9.7 below. Confidence bands are standard error (68%) for legibility.

Table 9.7: F_1-scores over varying methods and training set sizes for Vive Quaternion.

Method	T=1		T=2		T=3		T=4		T=5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Baseline	0.86	0.05	0.88	0.04	0.90	0.03	0.91	0.03	0.92	0.02
Mincer	0.84	0.05	0.86	0.05	0.88	0.05	0.89	0.05	0.90	0.05
3σ-per-class	0.79	0.11	0.85	0.07	0.87	0.05	0.89	0.04	0.90	0.04
3σ	0.77	0.06	0.74	0.08	0.75	0.10	0.72	0.13	0.72	0.14
Minimum Dist	0.74	0.18	0.79	0.15	0.81	0.15	0.82	0.16	0.82	0.16
Splice	0.74	0.05	0.72	0.09	0.69	0.10	0.67	0.10	0.65	0.11
Average Dist	0.80	0.04	0.79	0.05	0.77	0.06	0.75	0.06	0.74	0.07

Table 9.8: Percentage increase in error rates over baseline for Vive Quaternion.

Method	T=1	T=2	T=3	T=4	T=5
Mincer	10	17	21	26	27
3σ-per-class	44	28	30	24	21
3σ	60	125	159	216	243
Minimum Dist	83	81	92	110	122
Splice	77	144	217	273	319
Average Dist	37	85	135	181	210

Table 9.9: Vive Quaternion accuracy post-hoc results using the exact, two-tailed Wilcoxon signed rank test for top four results. Note, Friedman testing revealed a significant difference between methods, $\left(\chi_3^2 = 19.9, p < .001\right)$.

Pair	$W+$	p	r
Baseline - 3σ	0	$< .05$	1.0 L
Baseline - 3σ-per-class	0	$< .05$	1.0 L
Baseline - Mincer	0	$< .05$	1.0 L
3σ - 3σ-per-class	50	$< .05$	0.8 L
3σ - Mincer	52	$< .05$	0.9 L
3σ-per-class - Mincer	51	$< .05$	0.9 L

9.4.5.4 Mouse

Mouse accuracy results are shown in Figure 9.10 and Table 9.10. Baseline, 3σ, 3σ-per-class, and Mincer are all again statistically different from each other (Table 9.12). Like with Kinect, we find that Baseline and Mincer achieve high accuracy ($\geq 90\%$) using a single training sample per gesture class and they both improve as the training set size increases. Also like before, performance plateaus at $T = 3$. The difference in error rates between these two methods is low (Table 9.11). The remaining techniques do not perform as well—none achieve high accuracy.

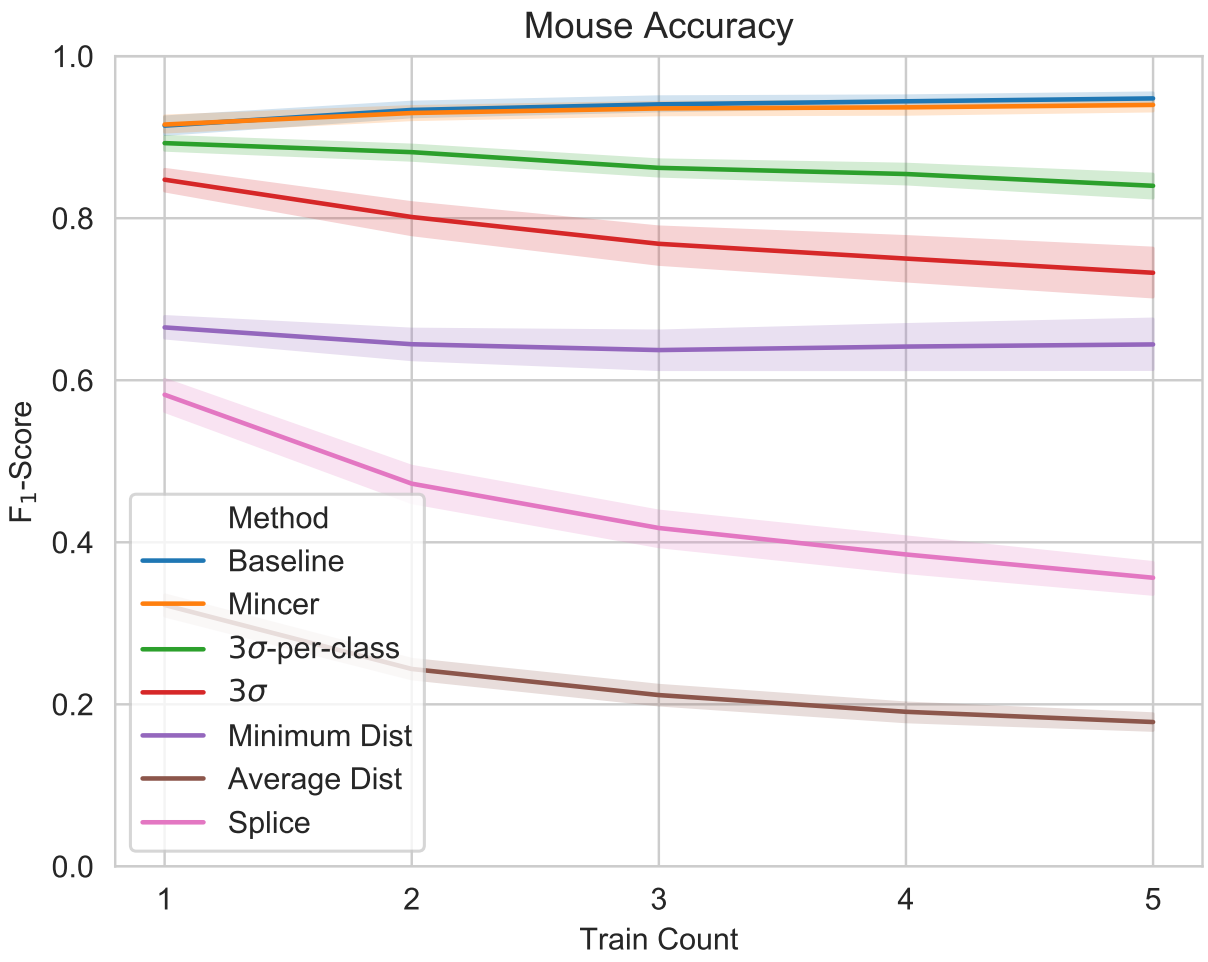


Figure 9.10: F_1-score over varying methods and training set sizes for Mouse. Exact values are given in Table 9.10 below. Confidence bands are standard error (68%) for legibility.

Table 9.10: F_1-scores over varying methods and training set sizes for Mouse.

Method	T=1		T=2		T=3		T=4		T=5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Baseline	0.91	0.03	0.93	0.03	0.94	0.03	0.94	0.02	0.95	0.02
Mincer	0.92	0.03	0.93	0.03	0.94	0.02	0.94	0.03	0.94	0.02
3σ-per-class	0.89	0.03	0.88	0.03	0.86	0.03	0.85	0.04	0.84	0.05
3σ	0.85	0.04	0.80	0.06	0.77	0.08	0.75	0.09	0.73	0.09
Minimum Dist	0.67	0.04	0.64	0.06	0.64	0.08	0.64	0.09	0.64	0.10
Splice	0.58	0.07	0.47	0.07	0.42	0.07	0.38	0.07	0.36	0.06
Average Dist	0.32	0.04	0.24	0.04	0.21	0.04	0.19	0.03	0.18	0.03

Table 9.11: Percentage increase in error rates over baseline for Mouse.

Method	T=1	T=2	T=3	T=4	T=5
Mincer	-1	5	8	13	15
3σ-per-class	25	78	131	161	206
3σ	77	199	289	348	412
Minimum Dist	290	435	509	543	582
Splice	387	694	878	1003	1134
Average Dist	690	1039	1225	1352	1476

Table 9.12: Mouse accuracy post-hoc results using the exact, two-tailed Wilcoxon signed rank test for top four results. Note, Friedman testing revealed a significant difference between methods, $\left(\chi_3^2 = 27.5, p < .001\right)$.

Pair	$W+$	p	r
Baseline - 3σ	0	$< .05$	1.0 L
Baseline - 3σ-per-class	0	$< .05$	1.0 L
Baseline - Mincer	0	$< .05$	1.0 L
3σ - 3σ-per-class	50	$< .05$	0.8 L
3σ - Mincer	52	$< .05$	0.9 L
3σ-per-class - Mincer	51	$< .05$	0.9 L

From the above results, we see that VKM using GPSR and Mincer yield high accuracy results that are always near optimal. Using this information we define The Dollar General as a continuous custom gesture recognizer that uses The Voight-Kampff Machine to decide rejection thresholds.

9.5 Discussion

It is clear from our evaluation that The Voight-Kampff Machine as part of The Dollar General pipeline performs very well, achieving near optimal performance in every test involving dynamic gestures. Among six threshold selection techniques, GPSR with Mincer specifically out performed all other selection techniques and was the only method to achieve high accuracy ($\leq 90\%$) across all four high-activity datasets. 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 to support gesture customization for a variety of input device types. In the next chapter, we engage in further evaluations of TDG, for which we reserve additional comments. We instead turn our attention to known VKM limitations.

9.5.1 Limitations and Future Work

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 well beyond the straightforwardness constraints we place on this work, but it is 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 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 (Algorithm B.21). 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 training set size. Differences in variability cannot always be known in advance but is highly relevant since ignoring

these differences can led to VKM underestimating the rejecting threshold, which in turns 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.

9.6 Conclusion

In this chapter, we 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 splicing generates realistic negative score distribution, but rejection threshold selection using arbitrary negative data did not lead to optimal results in our framework. 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 reveled that TDG with VKM is able to achieve not only high accuracy, but also near optimal performance (given Machete's ability to segment incoming data and Jackknife's ability to classify the gesture candidates). We believe TDG in general and VKM specifically 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.

CHAPTER 10: A FURTHER EVALUATION OF THE DOLLAR GENERAL

In the future...
Will I need my mind?

Or will Google...
Take over this grind?

Kongos
Autocorrect

We have now defined all components of The Dollar General (TDG) continuous custom gesture recognition pipeline. This includes Pitch Pipe (Chapter 4) for low-pass filter tuning, Machete (Chapter 7) for segmentation, Jackknife (Chapter 6) for robust measurement, and The Voight-Kampff Machine (Chapter 9) for rejection threshold selection. Although we have evaluated each component separately, we have yet to compare the entire pipeline against alternative techniques, which is the purpose of this chapter. Herein, we put TDG in competition with several state-of-the-art techniques to better understand TDG's capabilities and limitations. In addition, through a small scale user study, we also evaluate whether VKM comprises truly accessible and easy to implement techniques.

10.1 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 [32]. The competition organizers collected continuous data from participants

embedding gesture and non-gesture interactions. 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" [32, p. 94]. While interacting with the environment, participants were asked to perform one of five gestures at various times: cross, V-mark, caret, squre, 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 the 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 [128]. 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 10.1. 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$_2$

Table 10.1: SHREC 2019 compeition results reported as percentage values. Each row sums to 100%. See [32] 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

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.

10.2 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.

10.2.1 uDeepGRU

uDeepGRU is based on DeepGRU [155], 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 [44] for its recurrent layers, and a fully connected layer for its classification network. Details are available in [32].

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 dataset contains only segmented gesture data. 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.

10.2.2 Method and Results

We used an identical test protocol to that reported in our first evaluation; see Chapter 9.4.3. One difference is that we used all available training samples ($T = 5$), where in each iteration, we used

Table 10.2: uDeepGRU Kinect results

Method	T=1		T=2		T=3		T=4		T=5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Baseline	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

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 to disk. We say a user gesticulated if uDeepGRU outputs identical 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 10.2. 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 ($\approx 9\%$).

10.3 ChaLearn Looking at People 2014 Challenge

In 2014, ChaLearn[1] organized a competition for multimodal, continuous gesture recognition using data acquired from an RGB-D camera [67]. Their dataset comprises video sequences of users communicating with Italian sign language, though other actions are present in the data. In total, there are 20 sign language gestures collected from several participants that aggregate to more than 14,000 gesture instances. Being a multimodal competition, the organizers provide RGB, depth, and

[1] http://gesture.chalearn.org/

skeleton video data. The dataset is divided into annotated training and validation data. Additional details can be found on their website[2]. We expect to perform poorly with this dataset, because although it involves Kinect skeleton data, many gestures are static and involve specific hand poses, which Kinect does not track well. We included this test to validate our expectations.

Rather than measure F_1-score, this competition measures the Jaccard Index, which is the ratio of correctly labeled frames over the combined ground truth and false positives labels per gesture instance. All results are averaged into a single ratio. Using the competition's protocol, we collected results for a subset of the competition data. Whereas the competitors's accuracies varied between 0.27 to 0.85, TGD only achieves 0.15. This result confirms our expectation that TDG does not work well for static, pose-based gestures.

10.4 Evaluation of Implementation Straightforwardness

In this section, we discuss an informal, small scale exploratory user study with two participants, one with and one without machine learning experience. We had two goals. First, we wanted to ensure that one could implement VKM without difficulty should one need to do so, regardless of their background. Recall, a design objective of TDG is accessibility, which means that one is able to implement and debug our methods with appropriate reference materials. Second, we wanted to understand in what ways we could improve our presentation so as to accelerate development, which requires both expert and non-expert feedback.

[2] http://chalearnlap.cvc.uab.es/dataset/13/description/

10.4.1 Method

We designed a Python-based system that implements a gesture dataset loader, evaluation module, and recognizer. Our evaluation module implemented the user-dependent test protocol discussed in Chapter 9.4.3, and the recognizer was Jackknife. We designed the system so that upon execution, the evaluation module would load our high-activity Kinect dataset, filter the data with a central moving average filter, and train Jackknife with one random sample per gesture class. After loading the templates, our system would then call a method in Jackknife to learn a rejection threshold, which called into a skeleton VKM-based module to be filled out by each participant. In more detail, we provided skeleton method declarations for Select-Rejection-Threshold (Algorithm B.19), Estimate-Rejection-Threshold (Algorithm B.20), GPSR (Algorithm B.13), Optimal-*N* (Algorithm B.24), and Mincer (Algorithm B.23). We did not require that participants implement Estimate-Adjustment (Algorithm B.21) since we were concerned about time and, as discussed in Chapter 9.1.2, our intention is to provide look up tables for common values.

10.4.2 Procedure

We gave each participant a short document that describes template matching and VKM, our incomplete source code, and the associated VKM pseudocode. We then asked them to implement each method while being cognizant of which areas were most difficult to follow, and time how long it took to complete the assignment. No instructions were given to participants to understand the methods. We believe this is a typical use case where one requires gesture recognition support and is willing to implement the technique, but he or she does not need to fully understand the underlying mechanisms. Once they completed their task, they sent us a short write up on their experiences.

10.4.3 Results

The first participant was a 20 year old male junior in the Computer Science program at the Purdue University. He had no prior machine learning experience but had taken Calculus I and Statistics. He completed the assignment in approximately 3.5 hours with a half hour spent learning the materials and studying the code, two hours implementing the pseudocode, and one hour gaining a better understanding of the code while debugging. The participant wrote that understanding template matching was the most challenging aspect of the assignment, in part because we did not provide a sufficient explanation of dynamic time warping. GPSR and Mincer were, however, easier to understand, and although the participant did not fully grasp the code, he had no significant issues implementing all of the techniques from the given materials. On further inquiry, the participant believed that additional high level comments would aid in understanding. The participant summarizes their experience: "Approaching the objective without any background knowledge in recognizer algorithms[,] I had expected this to be more challenging than it turned out. [...] Overall, I was surprised that I could get the recognizer working with only a general understanding of the concepts and the pseudocode to translate."

The second participant was a 22 year old male senior in the Computer Science program at the University of Central Florida. He had extensive machine learning experience being a Google certified TensorFlow developer who has implemented various machine learning techniques in Matlab, including linear and logistic regression, and who is currently working with neural networks. Given the participant's background, he was able to offer different perspectives on the work. The participant wrote that although he had no experience with gestures or gesture recognition, he found the theory easy to understand and thought it was particularly interesting how we were generating negative data. He spent six hours completing the assignment, though a significant portion of this time was spent understanding the software architecture, as he appeared to dive in deeper than the

previous participant. One issue was that he expected a greater correspondence between our pseudocode and software for data access methods, *e.g.*, for accessing Jackknife's stored templates, and so spent time better understanding precise data representations. These were his only criticisms, and despite having access to advanced techniques given his background, the participant said he could see himself using Jackknife with VKM if his work required gesture based interactions, especially for VR-based applications and low-resource devices such as Raspberry PI.

10.5 Discussion

In Chapter 9, we found that TDG performed very well across all our four high-activity datasets: Kinect, Vive position, Vive quaternion, and Mouse. We now find that TDG also performs very well on continuous high-activity hand gesture data collected with a Leap Motion input device. Specifically, TDG outperformed all competitors in the SHRECC'19 competition, being the only system to achieve high accuracy. 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" [32, p. 98]. Although TDG has tunable parameters throughout the system (see Chapters 6 and 7), there is little effort involved in selecting appropriate values.

It is worth noting that it took significant effort for us to make uDeepGRU work on Kinect data, approximately two weeks working on and off. We had to learn the software architecture well enough that we were able to modify its synthetic data generation process (so that uDeepGRU would work well with high-activity Kinect data). We saw that performance improved with synthetic data

quantity, but we quickly hit GPU memory limits that restricted our use of synthetic data. However, with more time we believe we could make future improvements and achieve higher accuracy, but because TDG already works very well, we see no reason to expend this effort on alternative techniques in general, or uDeepGRU specifically.

We also note 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.

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; 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.

10.5.1 Straightforwardness of VKM

Our informal exploratory study revealed some interesting insights. First, evidence so far suggests that implementing VKM from pseudocode is neither challenging nor time consuming, which we see as an advantage of our approach. Rejection threshold selection is a challenging problem, and our methods appear to enable inexperienced developers as well as those familiar with machine

learning but not gesture recognition. Second, it may also be that dynamic time warping requires extra effort to understand compared to GPSR and Mincer, which is unsurprising. We admittedly did not try to explain DTW in detail, as we expected that participants would focus on VKM.

Based on our experiences in providing a scaffolded system in combination with participant feedback, we recommend the following:

1. *Descriptive Comments*: Conference paper pseudocode is often written in a way that consumes little space. There is an assumption that one will first thoroughly read the paper, and, thereafter, implement associated methods. However, we believe a common use case is that developers will simply skim a paper to gain an overview. In this case, it will be beneficial to include high level comments that describe the process. In this way, pseudocode can standalone and reduce cross referencing when one requires a deeper understanding as one implements an algorithm.

2. *Verification Techniques*: Pseudocode often comprise a number of methods that stand alone. This means that one can potentially unit test each method in isolation given the appropriate guidance, rather than debug the entire system after implementing it from pseudcode. To accelerate development, it will be useful for researchers to provide verification techniques. For instance, 2D step function, caret, and square trajectories can be hardcoded as $[(0,1),(1,1),(1,0),(2,0)]$, $[(0,0),(1,1),(2,0)]$, and $[(0,0),(0,1),(1,1),(1,0),(0,0)]$, respectively. Using these simple shapes, we can provide expected interclass dissimilarity measures, expected within class measures of synthetically generated samples using GPSR and Mincer, and expected rejection threshold values. With this data in hand, one can quickly implement and validate each method in turn, which will likely reduce debug and development time.

10.5.2 Limitations and Future Work

Most TDG limitations have been discussed in prior Chapters. However, we like the ChaLearn 2014 dataset because it exposes some limitations of our system and reveals where future work is needed. The challenging aspects of this competition are that it requires user independent recognition, several of the sign language gestures are static poses, and some gestures depend on hand orientation, which is unreliably tracked in the skeleton data videos. We expected to perform poorly since TDG was designed for dynamic gestures and we are unable to track hands, but it does makes us wonder if we can do more with respect to handling temporal data and static poses.

10.6 Conclusion

In this chapter, we further evaluated The Dollar General continuous custom gesture recognition pipeline. We found that TDG outperformed state-of-the-art deep learning based recognizers on continuous Leap Motion data collected for SHREC 2019. Based on competition results, we selected the second best system, uDeepGRU, and modified it to work with our high-activity Kinect data. We found that TDG, again, significantly outperformed its competition. However, we also found that TDG is not perfect. Our ChaLearn 2014 evaluation confirmed known Dollar General limitations—static and temporal-sensitive patterns are difficult for our system to detect. Finally, we found through a small scale user study that VKM is relatively accessible and easy to implement. In general, results from this chapter reaffirm findings from previous chapters, that the individual components are intuitive and achieve high, state-of-the-art accuracy.

CHAPTER 11: PROTOTYPE SELECTION[1]

I'm digging with my fingertips
I'm ripping at the ground I stand upon
I'm searching for fragile bones
(Evolution)

Korn
Evolution

It is possible to acquire a large dataset over time. One can collect samples from friends, family, coworkers, and volunteers; save query samples to a local repository; select samples from public dataset (such as [9, 147, 257, 267]); or generate synthetic samples from an online service [141]. However, due to the nearest neighbor pattern matching scheme use by $-family recognizers and The Dollar General, each additional training sample increases recognition processing time and storage requirements, both of which we strive to minimize. Therefore, when presented with a large dataset, it is beneficial to train with only those samples that will have the greatest impact on recognition accuracy, which is known as the prototype or instance selection problem [81]. Most prototype selection solutions are slow, complex, or have high coding overhead, all of which are contrary to our Dollar General design goals. One exception is the set of stochastic methods [163, 183, 224] that use intelligence random selection to iteratively improve the training set. Specifically, in this chapter we look at genetic algorithms and hill climbing, which are straightforward, efficient techniques that typically achieve high accuracy. We demonstrate that both perform well in a gesture customization context, though hill climbing specifically is a better choice because of its simplicity.

---

[1]This chapter contains previously published material adapted from the following article: Pittman*, Corey, Eugene M. Taranta II*, and Joseph J. LaViola Jr. "A $-family friendly approach to prototype selection." Proceedings of the 21st International Conference on Intelligent User Interfaces. 2016. *Equal Contributors. See Appendix C for associated copyright information.

11.1 Genetic Algorithm

Genetic algorithms evaluate the fitness of a population comprising multiple candidate solutions over a number of generations. At the end of each generation iteration, children solutions are generated from those candidates in the present generation that are most fit. Genetic algorithms leverage mutation and crossover operators to facilitate the exploration of new areas within the associated search space.

In the context of prototype selection, each candidate solution is a subset of all training samples stored in a database. During initialization, we generate a number of candidate random samples, where each candidate is a unique training set, and an independent recognizer is trained for each candidate. To evaluate the fitness of a given candidate, we draw a new random sample, perform classification with its associated recognizer, and record its accuracy. This operation is carried out for each candidate in the population. Thereafter, the most fit candidates are carried over into the next generation, while the remaining candidates are replaced with mutated children. To generate a mutated child, we select a carried over candidate and randomly replace a subset of training samples, or we combine two random carried over candidates into a single new training set. We repeat this process until we reach a maximum iteration count or a candidate solution achieve perfect recognition. In our implementation, we specifically used single point mutation and uniform crossover. We also used elitism to carry over the top 50% of each generation.

11.2 Random Mutation Hill Climb

Of all the methods we considered, Skalak's [224] approach to prototype selection using *random mutation hill climb* (RMHC) is perhaps the most appropriate technique with respect to coding overhead, complexity, understandability, speed, and accuracy. In his initial formulation, prototypes

are represented as integers that index samples in a training set, and the selection is encoded into a single bit string. The bit string is randomly mutated one bit at a time. On each iteration, the fitness of the encoded prototype bit string is evaluated utilizing a 1-nearest neighbor classifier. If the fitness of the mutated string is greater than all previously identified solutions, the mutated string becomes the new best (fittest) solution. This process terminates after a predetermined, but low, number of iterations. Our approach is identical except that we store an array of integers that index samples in a training set, and we randomly mutate whole integers rather than individual bits in order to reduce coding complexity. However, there are also a number of optimizations that we employ to speedup the process.

There is no sense in considering a solution in which not every class is represented. When the target prototype count is low, for example, this situation is easy to encounter, and significantly many more mutations are required to achieve high accuracy as compare to the following alternative. We ensure that there is at least one prototype per gesture when performing a mutation. For each class we keep a counter of selected prototypes that represent the class. When a prototype is to be mutated, if its associated class count is one, we randomly select another prototype from within the same class. Otherwise, there is no restriction. Finally, if the fittest solution wins its second mutation round and has classified a sufficient number of consecutive samples correctly, the process terminates (currently this is 64 times the class count).

We operate under a similar assumption as Skalak: that $-family recognizers are typically employed in problem domains where good prototypes appear in dense regions of the sample space. We also assume that most, if not all, gestures reside in a *sloppiness space* [85], so that rushed and sloppy instances are still distinguishable. Using these assumptions, instead of evaluating a candidate solution over the entire training dataset, we can randomly sample from the dataset. That is, samples are drawn with replacement and evaluated against the fittest solution and candidate solution, until either one solution clearly out performs the other, or a predetermine number of iterations pass

(we use 256 times the class count in our evaluation). Also, to ensure good test coverage, in each iteration we test one sample from each class. Since each result is a Bernoulli trial (either a sample is classified correctly or not), the sequence of all results is a Binomial distribution. Therefore we can use an exact Binomial test to compare the solutions.

The minimum variance unbiased estimator of the success rate is given by:

$$\hat{p} = \frac{x}{n}, \tag{11.1}$$

where x is the number of successes in a set of n Bernoulli trails. Let $\hat{p}_0$ be the success rate estimate of the fittest solution and $\hat{p}_c$ the estimate for a new candidate solution. As solutions are iteratively evaluated, we conduct a one-tailed hypothesis test to eliminate any candidate solutions that are unlikely to improve accuracy:

$$\begin{aligned} H_0 &: \hat{p}_c = \hat{p}_0 \\ H_1 &: \hat{p}_c < \hat{p}_0, \end{aligned} \tag{11.2}$$

where the null hypothesis is that the candidate solution is approximately as good as the fittest solution. Note, however, that the usual normal approximation to the binomial distribution is inappropriate because the success rate is often too high. For this reason we must conduct an exact test and manually find the critical value of the rejection region.

The cumulative distribution function (CDF) of the binomial distribution is as follows:

$$P(X \leq k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}, \tag{11.3}$$

where random variable X is the number of successes and p, again, is the probability of success. Given a level of significance, such as $\alpha = .05$, it is more efficient to solve the inverse problem: to

Table 11.1: Datasets used to evaluate selection methods. Additional details can be found in the associated references.

Name	Ref	Multistroke	Gestures	Participants	Total Samples
$1-GDS	[267]	No	16	10	4800
SIGN	[4]	No	17	20	33154
MMG	[9]	Yes	16	20	3200

find the critical value for the upper tail, $1-\alpha$, and calculate the summation backward from n down to k for $P(X > k)$. Why? First note that Equation 11.3 contains a binomial coefficient, which can be solved incrementally as the CDF is being calculated by using the following relation:

$$\binom{n}{k-1} = \binom{n}{k}\frac{k}{n-k+1}, \tag{11.4}$$

where k represents the number of successes in n trials, and the initial condition is $\binom{n}{k=n} = 1$. Second, note that because the success rate is high, the k containing $\alpha = .05$ of the lower tail (or equivalently the k containing $\alpha = .95$ of the upper tail) is close to n. For example, with $n = 1500$ and $p = .95$, then 5% of the distribution falls under $k = 1439$.

11.3 Evaluation

We ran a large scale evaluation to understand the effect of our proposed solutions on recognition error rates. We specifically compared our solutions against an optimal baseline recognizer trained with all available data and a pure random selection based recognizer. In this way, we could determine if there was an improvement over random selection and, if so, see how close to optimal we come. We chose to evaluate the proposed selection methods on the three datasets shown in Table 11.1.

Table 11.2: Absolute error rates (in %) for all datasets, selection modes, and recognizers for template counts $k = [1,5]$ and baseline using all templates ($k = n$).

Penny Pincher

	$1-GDS			**SIGN**			**MMG**		
k	GA	RM	Rand	GA	RM	Rand	GA	RM	Rand
1	2.0	1.5	8.8	4.8	4.1	14.1	23.4	22.6	37.8
2	0.9	0.6	4.8	2.4	1.5	8.8	11.3	8.7	25.5
3	0.5	0.5	3.4	2.0	1.1	6.0	8.6	4.5	20.2
4	0.4	0.5	2.5	1.8	0.9	4.5	6.3	3.2	16.2
5	0.2	0.5	2.3	1.3	0.9	4.2	5.2	2.4	14.0
n	0.0	0.0	0.0	0.4	0.4	0.4	0.8	0.8	0.8

Protractor

	$1-GDS			**SIGN**			**MMG**		
k	GA	RM	Rand	GA	RM	Rand	GA	RM	Rand
1	2.3	1.6	12.0	16.1	16.0	32.7	23.4	23.7	35.6
2	1.0	0.8	6.7	14.7	13.2	27.8	9.6	6.9	23.3
3	0.8	0.7	4.8	14.5	13.1	24.8	7.0	4.3	18.5
4	0.6	0.5	3.6	13.9	12.2	23.4	5.8	3.6	15.5
5	0.6	0.5	3.1	14.1	11.5	22.7	5.1	3.2	13.4
n	0.2	0.2	0.2	13.5	13.5	13.5	3.0	3.0	3.0

$N-Protractor

	$1-GDS			**SIGN**			**MMG**		
k	GA	RM	Rand	GA	RM	Rand	GA	RM	Rand
1	5.0	4.2	14.7	15.9	16.2	31.1	12.0	12.0	20.4
2	1.3	0.9	7.6	12.6	11.1	24.3	5.6	4.1	12.0
3	0.9	0.5	5.5	11.7	10.7	20.8	3.9	2.1	9.3
4	0.7	0.8	4.1	10.7	10.1	19.0	3.5	1.7	7.4
5	0.6	0.7	3.7	11.1	10.6	18.4	2.7	1.7	6.1
n	0.1	0.1	0.1	9.4	9.4	9.4	0.5	0.5	0.5

Table 11.3: Percentage reduction in memory and computation over each recognizer and dataset with five template loaded relative to baseline.

Recognizer	**$1-GDS**		**SIGN**		**MMG**	
	Mem	**Speed**	**Mem**	**Speed**	**Mem**	**Speed**
Penny Pincher	98.3	95.7	99.7	99.5	97.5	95.2
Protractor	98.3	97.7	99.7	99.7	97.5	96.8
$N-Protractor	98.3	97.4	99.7	99.6	97.5	97.7

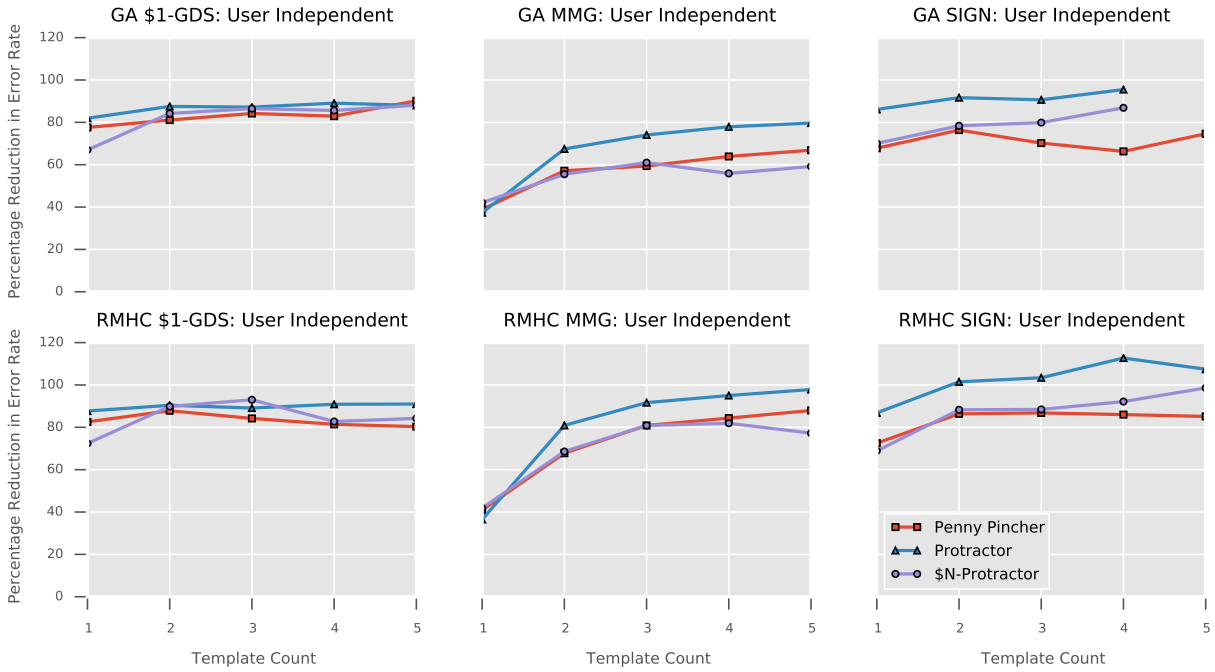


Figure 11.1: Percentage reduction in error rates relative to random selection error rates and baseline for different sample counts per gesture class for each dataset. The baseline is represented by 100% reduction in error rate and random selection is represented by 0%. Notice that the improvements are large for both selection methods.

11.3.1 Recognizers

We used a number of $-family recognizers to ensure that the results were consistent regardless of which recognizers were used:

Protractor. Protractor [143] is a speed-optimized version of the traditional $1 recognizer. This recognizer align the query and template gestures before measuring their inverse cosine distance.

$N-Protractor. $N-Protractor [9] is a speed-optimized version of the $N recognizer which leverages the optimizations proposed in Protractor to decrease the overall computation time of $N while maintaining similar error rates.

Penny Pincher. The Penny Pincher recognizer [238] is a reduced complexity recognizer that emphasizes speed and simplicity while maintaining high recognition rates. Penny Pincher reduces the matching process to addition and multiplication, allowing for significantly shorter matching times.

11.3.2 Procedure

In order to determine the effect of the reduced datasets on the error rates of the $-family recognizers, we randomly generated a number of tests for each combination of recognizer, dataset, and selection mode. The evaluations were user independent, so samples for each gesture type from each user were combined together. A prototype was selected from each gesture class within the selected dataset and was designated a candidate to be matched. The candidate was matched against the best reduced dataset generated using one of the selection methods. The candidate was not included in the gesture selection process for that particular test. Error rates were recorded for $k = [1,5]$, where $k * G$ is the number of prototypes to which the dataset should be reduced, and G is the number of distinct classes in the complete dataset. We repeated this process 500 times for each configuration.

11.3.3 Results

Results for the evaluation are shown in Table 11.2. For visualization purposes, we elected to report the percentage decrease in error rate in lieu of the absolute error rate because we are not comparing recognizers. We wanted to see how close to baseline error rates the selection algorithms were able to reach. These results are presented in Figure 11.1. Each of the recognizers improved dramatically over random selection in all configurations. We compared the RMHC and GA error distributions to random selection for each dataset, recognizer and level of k, which resulted in 90 pairwise comparisons. Using Welch's unequal variances t-test with a Bonferroni correction, all results were significant ($p < .05$). Error rates were reduced by over 70% in all cases for $1-GDS, 35% for SIGN,

and 25% for MMG. Table 11.3 also shows the reduction in time and space efficiency for k = 5. It is clear that reducing the size of the dataset reduces the computational load of the recognition process while still reaching similar recognition rates to baseline, according to Table 11.2.

11.4 Discussion and Conclusions

The results of the experiment point toward a clear benefit in using a supervised selection algorithm to reduce the number of prototypes while still maintaining only slightly worse error rates than when using the entire dataset. This idea is compounded in situations when there is a temporal constraint on recognition. Each of the configurations rapidly approached the baseline accuracy of the dataset for that particular recognizer.

Also notable is the fact that RMHC and GA performed similarly. Both methods are heavily influenced by the mutation genetic operator. Mutation is the primary method of exploration in GA with crossover being used to consolidate good candidates into potentially higher fitness individuals. RMHC is able to reach similarly optimal prototype subsets without the use of crossover or multiple candidates in each iteration. For this reason, we would recommend practitioners use RMHC in lieu of GA.

There are almost no downsides to carrying out this calculation, save the brief additional implementation time and time to carry out the preprocessing and selection, which is negligible for RMHC. However, we found that $N-Protractor was slow to carry out selection, particularly when working with a large, multistroke dataset (MMG). One possible way to solve this is to use a faster, more efficient recognizer like Penny Pincher during the selection process and then use the resultant subset of strokes to train a different $-family recognizer.

In this chapter we proposed two $-family principled prototype selection methods that use a iter-

ative stochastic search approach. Through an evaluation over multiple recognizers and datasets, we demonstrated that both techniques performed significantly better than random selection and achieved high recognition accuracy. However, random mutation hill climb was simpler to code and had similar execution times to our genetic algorithm. This makes RMHC a suitable choice for $-family related work and is complementary The Dollar General pipeline.

CHAPTER 12: ON ECOLOGICAL VALIDITY IN GESTURE DATA COLLECTION[1]

My body works, I know!
It's just the same, I know!
My only difference...
Is robot influence

King Gizzard & The Lizard Wizard
Robot Stop

Practitioners often collect test samples from participants using data collection techniques that allow one to focus solely on gesticulation and form [63, 235, 257, 267]. Such data is useful, enabling one to make relative comparisons between techniques and demonstrate improvements. However, it is unlikely that gestures produced in this way will capture variabilities present in form when one interacts with an application directly, making it difficult for researchers to understand how their efforts will translate into practice. Video games are one domain where such differences likely impact testing and experimentation. Prior research has shown that in-game gesture recognition accuracy is less than that of non-game data collected for training and testing within the same application [40, 236, 238]. Reasons for this drop are presently unknown, but one reasonable assumption is that gesture production variability increases with interaction complexity. For example, players may interact directly with virtual objects, and game-specific interaction requirements may alter gesture speed, size, orientation, and overall form. Understanding these differences will help inform practitioners on how to approach pattern recognition and user interface design for gestures.

---

[1]This chapter contains previously published material adapted from the following article: Taranta II, Eugene M., et al. "Moving Toward an Ecologically Valid Data Collection Protocol for 2D Gestures In Video Games." Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. 2020. See Appendix C for associated copyright information.

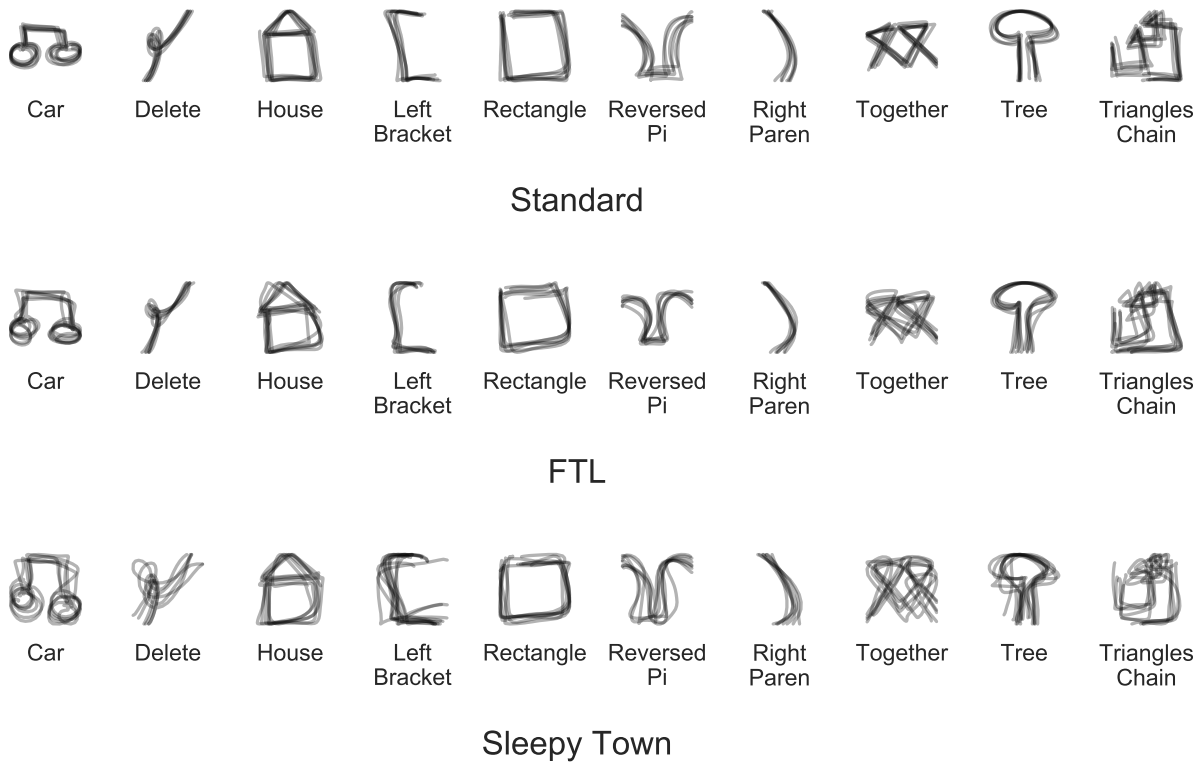


Figure 12.1: Gesture distributions elicited by three unique data collection applications from a single participant in our user study. Notice how form and varaibility differ signifcantly across the three conditions.

Toward this end, we developed three data collection applications: a standard application, a simple game called *Follow the Leader (FTL)*, and a complex game called *Sleepy Town*. Our standard application implements a typical data collection scenario. FTL introduces a trivial game play mechanic and is designed specifically for low implementation effort, enabling others to quickly adopt the protocol into their own work. Finally, Sleepy Town presents a top-down city view and allows for navigation as well as gesture interaction with non-playable characters. Through a within subjects experiment involving 18 participants, we quantify differences in gesture production between all conditions, confirming there does exist significant differences. We discuss our findings and their implications. Specifically, in this chapter we demonstrate that commonly employed data collection

practices inadequately capture gesture production variability relative to that found within practical applications generally and games specifically; and we describe an easy to implement protocol called *Follow the Leader* that increases gesture production variability, yet leverages technology already present in most data collection applications. Further, we used knowledge discovered through this study to inform our design of The Voight-Kampff Machine described in Chapter 9.

12.1 Data Collection Applications

We designed three data collection applications, one that replicates common practice and two that employ gestures in a game environment. We further developed all applications with Unity version 2019.1.11f1, a popular game engine designed for 2D, 3D, VR, and AR experiences. Each of these applications are described in this section.

12.1.1 Standard Data Collection

Our standard data collection application, shown in Figure 12.2, asks users to draw gestures that our software specifies one at a time. We horizontally center requests at the display's top, and users may gesticulate anywhere on the display. After one produces a gesture sample, two buttons appear that allow one to save or delete their sample. If satisfied, one can save their result and continue onto the next request, or delete their sample and produce a new variant. Gesture requests are purely random, as we provide no guard against consecutive identical requests. With this design, data collection is relatively stress free and comfortable. We impose no time pressure, feedback, or expectations so that users remain in full control.

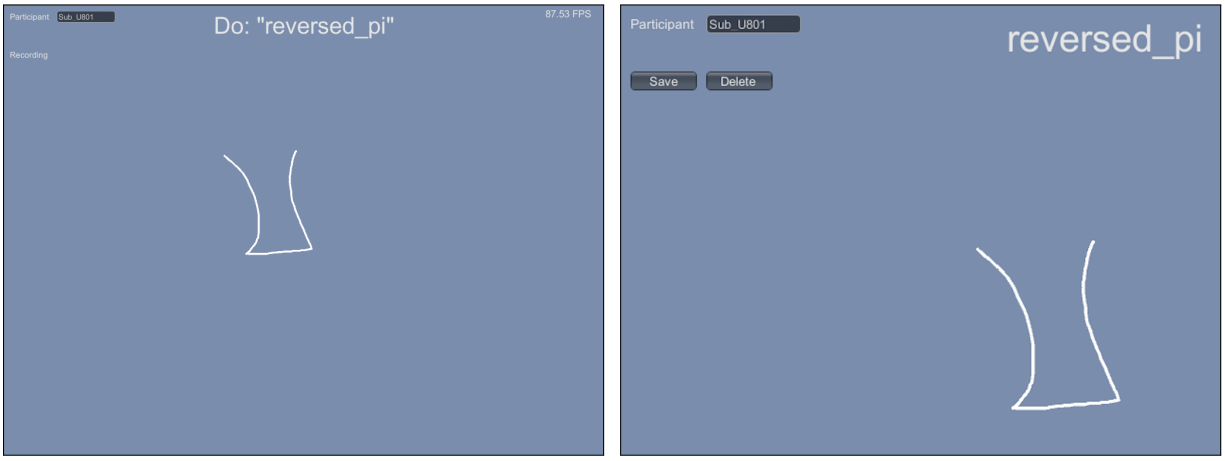


Figure 12.2: Standard data collection application: Left, user draws a gesture as requested above, i.e.,"reversed-pi." Right, two buttons appear that allow user to save or delete and retry.

12.1.2 Game: Follow the Leader (FTL)

Our first game is named after and inspired by a popular children's game called *Follow the Leader* (FTL), in which participants line up behind a leader whom all must follow and whose random actions they must exactly replicate. In a similar way, as presented in Figure 12.3, we present the leader as a series of seemingly random trajectories on one's display. A player must try to replicate precisely what they see, keeping pace with their leader in time and space. At random intervals, we further present a gesture request, whereby players must stop following, gesticulate anywhere on the screen, and return to following as quickly as possible. The intention of this design is to create a sense of urgency not present in standard data collection protocols.

Leader trajectories are prerecorded samples produced by the developer using our standard data collection application described above. All leader samples are intentionally arbitrary[2], being a random collection of geometric shapes, words, and scribbles (see Figure 12.3 for some examples). During game play, FTL randomly draws a single leader sample and replays its trajectory at a rate matching

[2]We attend to the intentionally arbitrary nature of our design later in the discussion.

Figure 12.3: Follow the Leader (FTL): Upper left, a leader (white) renders a random trajectory that the player (black) must replicate in realtime as closely as possible. At random intervals FTL makes a gesture request to which one must immediately attend as the leader continues on without pause (upper right), and to which one must return upon gesture completion. The lower panels each show ten randomly selected leader trajectories to illustrate type and variety.

its recorded speed. Once the sample is fully rendered, we repeat this process, continuously, until a predetermined number of gesture requests are made and satisfied.

12.1.2.1 Design Considerations

FTL is designed to be a middle ground between standard data collection and a fully featured video game. We expect an increase in gesture production variability as a player must attend to multiple dynamic tasks. FTL is further motivated by our desire to minimize developer effort should one decide to use an FTL approach in their own data collection work. Specifically, our goal was to increase variability by leveraging machinery already present in most data collection tools such as to display, capture, and randomize gestures. To use FTL, one will first define a gesture dataset, render leader sequences in random order, ask participants to follow leader actions as closely as possible, periodically make gesture request that participants immediately perform, and continue as such until all training data is collected.

12.1.3 Game: Sleepy Town

Our second game, Sleepy Town, offers an entirely unique experience relative to FTL by introducing comparatively greater game play complexity. In this fantasy-based game, the local government has decreed that its citizens must adopt an Uberman sleep schedule[3] so as to become a more productive town. One week after this law has gone into effect, Leopold happens across the city, where he finds that its citizens appear to be wandering aimlessly about without purpose. Further, in a state of sleep deprived delirium (because there could be no other reason), its citizens pursue and attack outsiders on sight. Luckily, Leopold is a magician who knows a variety of sleeping spells.

In Figure 12.4, we present the player's top-down view of a simple city scene along with an illustration of its two game play mechanics. First, to navigate through Sleepy Town, one sketches a path rendered in red that begins from within Leopold's halo. He will immediately begin to run

[3] A polyphasic sleep schedule in which individuals nap for 20 minutes at equidistant intervals throughout the day, usually six times per day. This schedule is notoriously difficult to adopt.

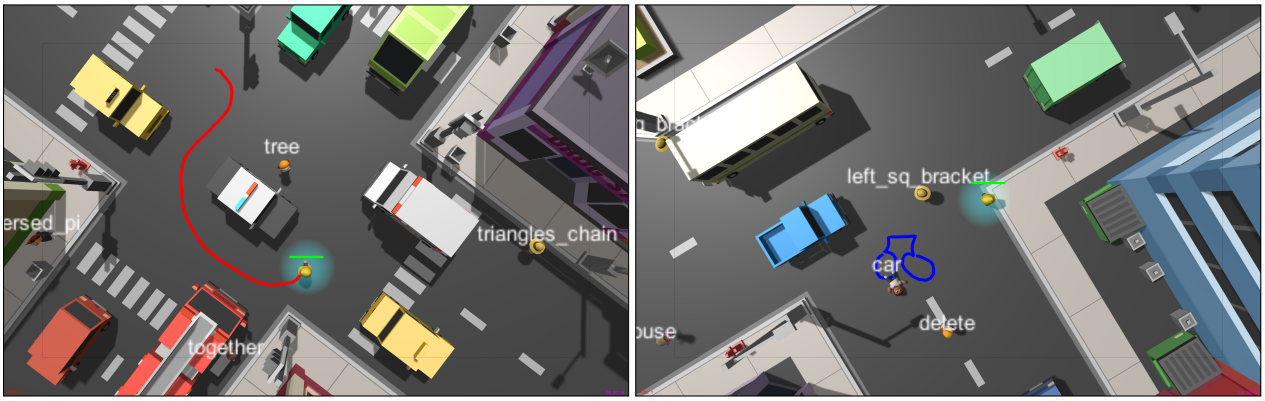


Figure 12.4: Sleepy Town: Left, player sketches a path that Leopold follows through the scene. Right, player draws a car gesture that causes an NPC to fall asleep as he crosses over its threshold.

along the player's sketched route until he reaches the end or until the player constructs a new route. As one approaches their display's physical boundary, we automatically rotate the camera around their point of contact in a way that allows him or her to continue sketching a continuous path; and when not routing, the camera automatically follows Leopold as he traverses his assigned route. When one instead sketches outside of his halo, our game transitions to gesture mode. Specifically, we project the player's input into the environment, rendering blue spell strokes. Above each non-player character (NPC) is written a gesture name—the spell that puts him or her to sleep. Once drawn, if an NPC walks over this gesture, he or she will immediately collapse into some much needed slumber. Like before, gesture assignments are randomized, and players continue to evade or put citizens to sleep until we collect a predetermined number of samples from each gesture class.

12.1.3.1 Design Considerations

In designing Sleepy Town, we were concerned with providing a realistic game play experience to ensure ecological validity. While a game can take on almost any form, a common design approach is to mimic elements found among successful games [2]. In this regard, we provide a clear

objective, visual feedback, interactive elements, risk, reward, and variability through navigation, gestures, and health pack pickups. Further, Sleepy Town adheres to the playability heuristics described by Desurvive *et al.* [52] and Pinelle *et al.*'s usability principles [189]. For example, the camera is never obscured. Relevant game state information is always presented in the form of an overlay with health information. Enemy behavior and user movement are similarly consistent and were found to be fair by users. In this way, we ensure an ecologically valid game play experience. Further, Sleepy Town's design is inspired by prior work [40, 236]. In our design, we ensure that gestures are also spatially relevant and require interaction with virtual objects. To illustrate, gesticulation requires that players align their gestures to intercept citizens along their trajectory. We believe this can influence shape, time, size, and sloppiness. And depending on the relationship between citizen, camera, and environment, the ability of a player to intercept a citizen will vary. Although we did not collect player experience metrics, we anecdotally received unsolicited positive feedback and participant requests for us to publish our game on an app store, which suggests that we provided a compelling gameplay experience.

12.2 Performance Measures

To understand gesture production differences between data collection applications, we employ a variety of performance measures: relative, global, and coverage. Each type offers a unique perspective on how users gesticulate within a given environment, which we discuss in this section.

Relative Accuracy Measures: One set of twelve designed by Vatavu *et al.*[251] for stroke gestures are the so-called relative measures. Given a gesture set, one first selects a representative task axis, typically the distribution's centroid. One then measures each remaining sample against this axis to understand how gestures vary within the random sample relative to a canonical form. An example

Table 12.1: Subset of the relative measures defined by Vatavu *et al.* [251] that we use in this work.

Name	Abbr	Description
Shape Error	ShE	Average difference between corresponding points
Shape Variability	ShV	Standard deviation of shape error differences
Length Error	LE	Sum of differences between gesture arc-lengths across corresponding points
Size Error	SzE	Difference in bounding box sizes
Bending Error	BE	Average of differences between turning angle at corresponding points
Bending Variability	BV	Standard deviation of turning angle difference between corresponding points
Time Error	TE	Difference in gesture production time
Time Variability	TV	Standard deviation of time difference between corresponding points
Speed Error	VE	Average difference in speed between corresponding points
Speed Variability	VV	Standard difference between differences in speed

relative measure referred to as the shape error follows:

$$\mathrm{ShE}(p) = \frac{1}{n}\sum_{i=1}^{n} \| p_{\sigma(i)} - \bar{p}_i \|, \tag{12.1}$$

where p is a stroke uniformly resampled to n points, $\bar{p}$ is the similarly resampled task axis, σ is a permutation function that aligns points in p with points from $\bar{p}$. In words, ShE measures the average difference between corresponding points. A brief description of each relative measure is given in Table 12.1, though the associated mathematics are omitted (see [251] for more information). In addition to the ten listed relative measures, we also report their geometric mean (Mean Measure) as a summary of error and variance across all measures, enabling one to quickly see the aggregate effect between conditions.

One issue with the relative measures in their specified form is that they do not allow for a direct comparison between distributions collected by unique protocols and/or hardware. For example, when one collects a distribution of samples with a large display compared to that of a smaller display, then the Euclidean distance-based shape error results will report a larger dispersion in the large display condition compare to that collected with the smaller display apparatus. For this

reason, we z-score normalize both position and time data for all samples residing within the same distribution. Specifically, we measure the bounding box size for all samples within a distribution and subsequently rescale all samples by the largest z-score normalized extent, so as to preserve aspect ratio. This normalization step allows us to directly compare the intra-distribution dispersion of those measures reported in Table 12.1 when collected amongst different data collection devices and protocols. We refer to these as *scaled* relative measures.

Global Measures: In addition to those relative measures just discussed, we also collect and report on a variety of absolute global measures. Namely, we examine the bounding box area, path length, gesture production time, and indicative angle variance, which are classic measures commonly used in gesture production analysis [215]. In our context, bounding box area informs one about the size of gesticulations across protocols as does path length. Differences in size and length force one to consider possible explanations for why users choose to vary their size with respect to the given task and apparatus. Gesture production time provides insight into how hurriedly a population produces gestures under a given condition, and variation in the indicate angle gives insight into orientation consistency under the same conditions. This latter measure is especially important given that a recent trend in recognizer research has been to drop rotation invariance [235, 238, 248, 250, 253].

Coverage Measures: Both relative and global measures yield important information on dispersion, yet fail to provide insight on form differences between distributions. For instance, although two unique random samples are identically self similar according to a given relative measure, this does not guarantee that their shapes are similarly identical, which directly impacts a recognizer's ability to match patterns. For this reason, we also report coverage via the modified Hausdorff distance

[57], defined as:

$$H(\mathscr{A},\mathscr{B}) = \max\left(d\left(\mathscr{A},\mathscr{B}\right), d\left(\mathscr{B},\mathscr{A}\right)\right) \tag{12.2}$$
$$d(\mathscr{A},\mathscr{B}) = \frac{1}{\mid \mathscr{A} \mid} \sum_{a\in\mathscr{A}} \min_{b\in\mathscr{B}} \left(f(a,b)\right),$$

where $\mathscr{A}$ and $\mathscr{B}$ are independent gesture sets, and f is a dissimilarity measure. In words, we calculate the average dissimilarity of each sample in a first dataset to its nearest neighbor in the second dataset, and we do this again in the opposite direction, which yields two averages. The maximum of these average dissimilarities then gives us a sense of how well the distributions cover each other.

A number of recognizers utilize Euclidean Distance in their pattern matching approach. For this reason, we employ the squared Euclidean Distance over normalized samples as our primary dissimilarity measure (f in Equation 12.2). Similar to \$1 [267], we resample, scale, and rotate all samples before making our measurement to ensure we are comparing differences in underlying form.

12.3 User Study

We conducted an experiment to better understand differences in gesticulation between a standard data collection protocol and games using the applications described in our previous section. We designed our experiment to test the following hypotheses:

H1: Gesture production variability is application dependent.

H2: Standard practice yields the least amount of gesture production variability.

H3: Sleepy Town yields the most amount of gesture production variability.

12.3.1 Subjects and Apparatus

We recruited 18 participants (12 male and 6 females) from a local university, all were right handed, all had prior experience with touch-based electronic interfaces and 14 had prior experience with pen-based electronic interfaces. Further, the population's mean age was 20.3 years old and ranged from 18 to 28. The experiment duration ranged from 30 to 50 minutes, and each participant was compensated $10 for their time. For a pen-based interactive display we used a Wacom Cintiq Pro 16 with display size of 15.6 inches (37.62 cm) and resolution of 3840x2160 pixels (UHD). We used the pen (6 inch, 15.5 cm) included with Wacom device for data collection.

12.3.2 FTL: Implementation

To facilitate offline processing, we record which leader or gesture request command is displayed at each moment in time. When FTL makes a gesture request, we inform the participant, and once the associated gesture is complete, we press a key that logs the request as complete. Afterward, we use automation implementing Penny Pincher [238] to classify each stroke recorded during the session. We further visually confirm all results and manually correct any errors.

12.3.3 Sleepy Town: Implementation

Actions are fast and sporadic in this game environment, so it is not possible to classify strokes in real-time as users play our game. Instead, using key button presses, we count when we believe players produce certain gestures. Once we collect a sufficient sample count for each gesture based

on investigator key-press feedback, the game terminates and all strokes are saved to disk. In a post processing step, all strokes are classified and errors are manually corrected.

12.3.4 Procedure

We presented each participant with a consent form that explained our experimental procedure and informed him or her of their rights. We then gave each individual a pre-questionnaire so as to collect demographic information, after which we explained our research. Participants were next introduced to the ten gesture classes shown in Figure 12.1 and allowed to practice them on paper until they were satisfied with their performance, though a reference sheet was kept nearby throughout the entire session. Once comfortable, participants used each of the three data collection application in a counterbalanced order. For each application, we first introduced its mechanics to the participant and then allowed him or her to practice until they were confident. Thereafter, we recorded at least six samples of each gesture class. Finally, we asked participants to fill out the NASA Task Load Index (TLX) questionnaire upon completion of each data collection task so as to assess subjective workload.

We choose to use the ten gestures shown in Figure 12.1 because of their prevalence throughout the custom gesture recognition literature. Although there were many to choose from, these ten also vary in familiarity and difficulty, and have good separability, which facilitates the use of our offline post processing tools. Our choice to limit the gesture class count to ten was driven only by logistics. To produce a minimum of six samples per class over three conditions took the longest participants approximately one hour. We feared that more samples or classes would lead to fatigue or effortless gesticulation.

12.3.5 Design and Analysis

We choose a within-subject design for our experiment in order to compare writer-dependent gesture production variations across each of the three data collection applications. In this way, we had one independent variable, *application*, with three levels: Standard, FTL, and Town. Our dependent variables are the global, relative, and distribution coverage measures discussed in the previous section.

For each participant, per gesture, we first computed each measure. We then averaged together the individual gesture class results per participant, which provided us with eighteen responses per condition. We thereafter used Friedman omnibus testing to detect differences between treatments, and exact Wilcoxon signed-rank testing for post hoc analysis. Finally, we used the Holm–Bonferroni step down procedure to control family-wise error rates [100].

12.4 Results

12.4.1 Relative Measures

The results of the relative measure comparisons are shown in Figure 12.5. Results of our Friedman tests showed significant differences across all conditions for all metrics, with the results of the test being presented in Table 12.2. Post-hoc analysis of the relative metrics provided additional insight into the differences between different applications and their distributions. The results of pairwise comparisons are presented in Table 12.3.

We first note that the Shape Error and Shape Variability measures between all applications are significantly different, increasing from standard practice to FTL and again from FTL to Sleepy Town. This result indicates that the position of corresponding points in a normalized space after spatial re-

Table 12.2: Friedman test results for relative measures. All results were significant.

Name	Test Statistic and Significance
Shape Error	$\chi^2(2) = 25.33, p < 0.00001$
Shape Variability	$\chi^2(2) = 30.33, p < 0.00001$
Length Error	$\chi^2(2) = 14.88, p < 0.001$
Size Error	$\chi^2(2) = 10.11, p < 0.01$
Bending Error	$\chi^2(2) = 23.11, p < 0.00001$
Bending Variability	$\chi^2(2) = 22.33, p < 0.0001$
Time Error	$\chi^2(2) = 16.78, p < 0.001$
Time Variability	$\chi^2(2) = 25.44, p < 0.00001$
Speed Error	$\chi^2(2) = 19.44, p < 0.05$
Speed Variability	$\chi^2(2) = 23.18, p < 0.00001$

Table 12.3: Pairwise Wilcoxon signed rank test results for relative measures.

Name	Standard-FTL	Standard-Sleepy Town	FTL-Sleepy Town
Shape Error	$Z = -3.79, p < 0.001, r = 0.63$	$Z = -4.08, p < 0.0001, r = 0.68$	$Z = -3.79, p < 0.001, r = 0.63$
Shape Variability	$Z = -3.96, p < 0.0001, r = 0.66$	$Z = -3.96, p < 0.0001, r = 0.66$	$Z = -4.23, p < 0.0001, r = 0.66$
Length Error	$Z = -1.29, p = 0.20$	$Z = -3.37, p < 0.001, r = 0.56$	$Z = -3.01, p < 0.01, r = 0.50$
Size Error	$Z = -1.16, p = 0.25$	$Z = -3.16, p < 0.01, r = 0.53$	$Z = -2.65, p < 0.01, r = 0.44$
Bending Error	$Z = -2.33, p < 0.05, r = 0.39$	$Z = -4.08, p < 0.0001, r = 0.68$	$Z = -4.08, p < 0.0001, r = 0.68$
Bending Variability	$Z = -2.17, p < 0.05, r = 0.36$	$Z = -3.16, p < 0.01, r = 0.53$	$Z = -3.16, p < 0.01, r = 0.53$
Time Error	$Z = -0.212, p = 0.83$	$Z = -2.89, p < 0.01, r = 0.48$	$Z = -2.76, p < 0.01, r = 0.46$
Time Variability	$Z = -2.61, p < 0.001, r = 0.43$	$Z = -4.16, p < 0.0001, r = 0.69$	$Z = -4.23, p < 0.0001, r = 0.71$
Speed Error	$Z = -1.15, p = 0.25$	$Z = -4.08, p < 0.0001, r = 0.68$	$Z = -3.32, p < 0.001, r = 0.55$
Speed Variability	$Z = -0.45, p = 0.65$	$Z = -3.30, p < 0.001, r = 0.55$	$Z = -3.20, p < 0.01, r = 0.53$

sampling are less varied under standard data collection practices. We see similar trends in Bending Error and Bending Variability, which shows that the angles between corresponding point triplets yield increased curvature differences with increased game complexity. Size and Length Error are not significantly different between standard practice and FTL, but both differ from Sleepy Town, where we observe less consistency. That is the relative variation in size and temporal alignment between points is greatest in Sleepy Town. We see a similar result for Speed Error and Variability in that Sleepy Town exhibits the most error and variability. Finally, the Mean measure (Figure 12.6 left) clearly echos the individual relative measure results—standard practice gesture productions are most consistent and Sleepy Town least, leaving FTL in the middle.

12.4.2 Global Measures

Global measure comparisons results are shown in Figure 12.6. Results of our Friedman tests showed significant differences across conditions for Area ($\chi^2(2) = 20.11, p < 0.05$), Angle Variance ($\chi^2(2) = 25, p < 0.05$), Length ($\chi^2(2) = 21.78, p < 0.05$), and Duration ($\chi^2(2) = 27.44, p < 0.05$) only. Post-hoc analysis provided further insight about differences between the conditions. For Area, there was a difference between standard practice and Sleepy Town ($Z = 3.79, p < 0.001, r = 0.63$), as well as FTL and Sleepy Town ($Z = 4.08, p < 0.0001, r = 0.68$). For Angle Variance, there was a difference between standard practice and Sleepy Town ($Z = -3.87, p < 0.0001, r = 0.64$), as well as FTL and Sleepy Town ($Z = -2.95, p < 0.01, r = 0.49$) and FTL and standard practice ($Z = -2.41, p < 0.05, r = 0.40$). Finally, Duration showed a difference between standard practice and Sleepy Town ($Z = 4.23, p < 0.0001, r = 0.71$), as well as FTL and Sleepy Town ($Z = 2.12, p < 0.05, r = 0.35$) and FTL and standard practice ($Z = 4.23, p < 0.0001, r = 0.71$). Note that although FTL and Sleepy Town measure lower in Area and Duration, their relative measures show greater error and variability.

12.4.3 Coverage Measures

The results of the coverage measure comparisons are shown in Figure 12.7. Results of our Friedman tests showed significance both between conditions ($\chi^2(2) = 32.44, p < 0.05$), and within conditions ($\chi^2(2) = 36, p < 0.05$) for our coverage metric based on modified Hausdorff distance. Post hoc analysis shows significant differences between each pairwise comparison. For intra-condition distances, all pairwise combinations were equally significant ($Z = -4.23, p < 0.0001, r = 0.71$). For inter-condition distances, FTL-Sleepy Town vs Standard-FTL and Standard-FTL vs Standard-Sleepy Town were equally significant ($Z = -4.23, p < 0.0001, r = 0.71$), while FTL-Sleepy Town vs Standard-Sleepy Town was slightly less significant ($Z = -3.96, p < 0.0001, r = 0.66$).

12.4.4 Perceived Workload

To analyze perceived workload, we ran a Friedman test on the raw NASA TLX data collected from the users after completing each task [92]. The Standard collection tool showed lower workload ($M = 31.22, SD = 20.0$) than both Sleepy Town ($M = 48.44, SD = 23.7$) and FTL ($M = 80, SD = 25.0$). There was a significant difference between the three conditions ($\chi^2(2) = 23.07, p < 0.05$). Our post hoc analysis found that there were differences between Standard collection tool and FTL ($p < .0001$), Sleepy Town and FTL ($p < .001$), and Standard collection tool and Sleepy Town ($p < .0001$). The increased task load introduced by FTL did not introduce additional gesture variability, as seen in the global and relative measures.

12.5 Discussion

From our experiment, it is clear that data collection protocols influence gesture form, and by introducing even minor game play mechanics to a protocol, we can expect a significant increase in variability. Specifically, by analyzing global measures, we observe differences in production time, where both game conditions elicited faster responses relative to standard practice. Since speed has been correlated with recognition accuracy [251], and we find that participants gesticulate with greater haste under our alternative conditions, future data collection protocols should work to elicit speed variability.

Second, we found a significant difference in size and orientation variability between Sleepy Town and the remaining conditions. Interestingly, participants were as likely to draw large gestures when playing FTL as they were when using standard practice. However, in Sleepy Town, players were forced to interact with dynamic content in a timely manner. We believe that these targeted interactions had an influence on gesture size. For a similar reason, we visually observed that some

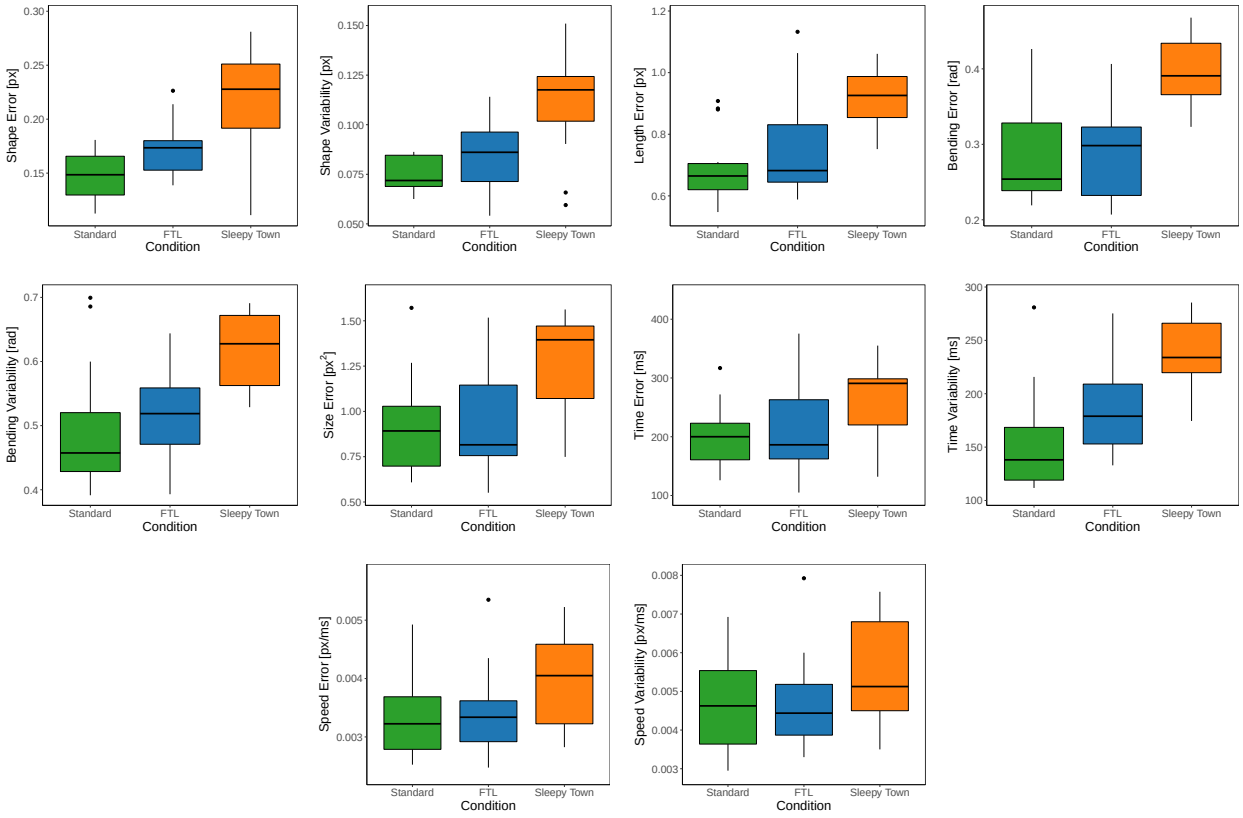


Figure 12.5: Collection of relative metrics. The top row, from left to right, includes Shape Error, Shape Variability, Length Error, and Bending Error. The middle row contains Bending Variability, Size Error, Time Error and Time Variability. The last row contains Speed Error and Speed Variability.

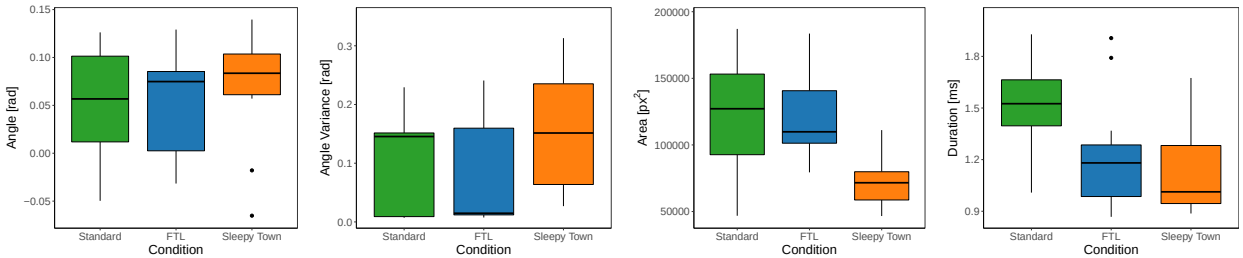


Figure 12.6: Collection of global metrics. From left to right, Indicative Angle, Indicative Angle Variance, Gesture Area, and Duration.

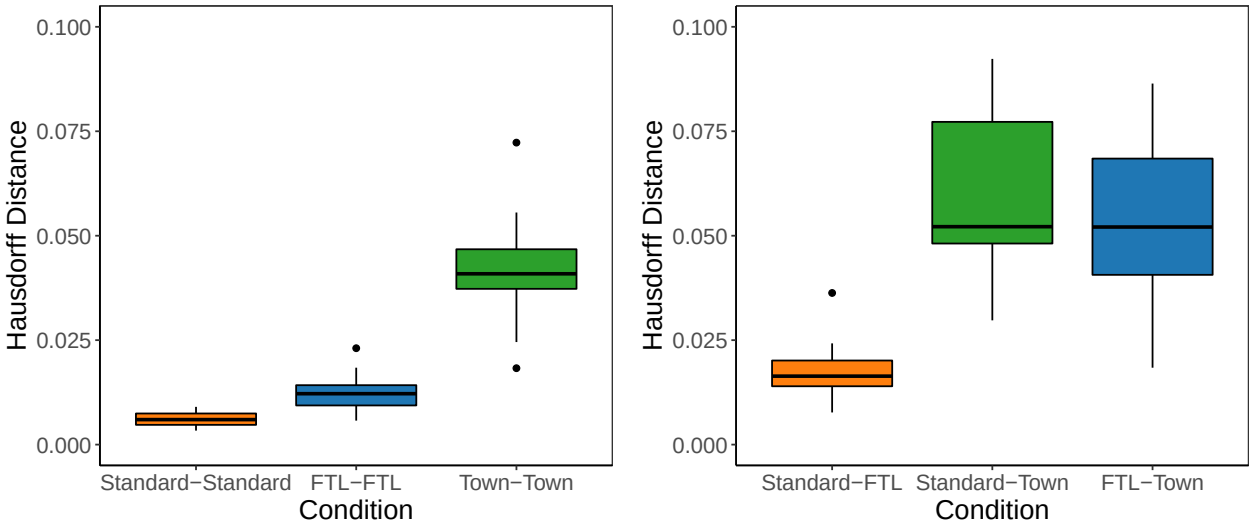


Figure 12.7: Results of comparisons betweeen gesture form from different conditions using Hausdorff Distance.

players oriented their gestures toward Sleepy Town citizens so as to intercept individuals along their apparent trajectories and to put spells on specific characters. Such orientation variability has an important effect on recognizer accuracy with respect to alignment. In recent years, custom gesture recognizers have dropped support for rotation invariance [235, 238, 248, 250, 253]; however, with this new data in hand, practitioners might consider again adopting rotation invariance, especially for circumstances that involve interactions with dynamic content. Intuitively, our findings also suggest that future data collection protocols should work to elicit variability in both orientation and size.

Relative accuracy measures also reveal a number of insights. Across all twelve measures analyzed, Sleepy Town elicited significantly higher variation compared to FTL and standard practice. Of particular interest are Shape Error and Variability, because like speed, these measures have been correlated with recognizer accuracy [251], where lower values lead to better performance. Specifically, these measures inform us about the average deviation and variation between corresponding

points of a given sample against the distribution's centroid, both of which are significantly higher in our game environments. In other words, we find that players are less consistent in their productions when reacting to dynamic content. Because standard practice does require such interactions, data captured with such tools are unlikely to exercise recognizers to the same extent. Bending error and variability tell a similar story.

Finally, coverage measures reinforce what we already learned from relative accuracy measure analysis—-standard practice yields a more consistent random sample. We find that the distance between nearest neighbors within the same distribution are furthest apart in the game environments, with Sleepy Town being greatest by a considerable margin. Consequently, standard practice data is unable to provide adequate coverage for either game environment. This finding is consistent with prior work, where researchers found accuracy drops in video games relative to tests conducted with training data [40, 236, 238].

Findings across all measures confirm our hypothesis that gesture production variability is application dependent, as across each condition, we observe unique variations in size, speed, orientation, and form. We further confirmed our second hypothesis that standard practice yields the most consistent sampling. And finally, we also confirmed that Sleepy Town, being the most complex game with respect to interaction style, elicits the most variable responses. Based on these findings, we recommend that user interface designs and pattern recognition researchers validate their work with data collected from within rather than outside of their target environment, or adopt new data collection protocols. Given that standard practice does not yield sufficient variability, results reported with such data represent optimistic upper bounds on performance rather than provide clear expectations.

12.5.1 About Follow the Leader

We designed FTL as a data collection application for low cost implementation effort in that many design choices were driven by practical logistical considerations. Consider that a typical standard application already has the ability to collect and render gesture data. Using this functionality, a practitioner can easily collect leader samples that their system replay while it displays text commands and collects new input. Subsequently, using any off-the-shelf recognizer such as $Q [253], one can post process their data to classify new input using designer made templates, after which one just manually corrects any minor miss-classification error. Inline with simplicity, FTL does not time, score, nor provide any user performance feedback, though one could if they so chose. Despite this apparent feature scarcity, FTL is still able to elicit highly variable responses relative to standard practice, and for these reasons, we believe FTL is a good starting point for future data collection efforts as we begin to move toward more ecologically valid protocols.

12.5.2 Future Directions

In this work, we focused on unistroke gestures of varying complexity to facilitate user study duration, but we intend to follow up with multistroke as well as hand and fully body gesture analysis. We additionally intend to explore additional FTL game mechanics that may yield even more variability, including orientation and timed response requirements. Last, we also believe FTL will be especially useful for high activity continuous data acquisition, as players are forced to maneuver between gesture and non-gesture interactions. In this way, FTL will capture challenging datasets that researchers and practitioners can use to test gesture segmentation and recognition.

12.6 Conclusion

We have presented results from a user study demonstrating that standard data collection protocols do not capture the true variability of gesticulation within a game environment. This result holds for even our simplest game, Follow the Leader (FTL), which yielded variation significantly different form that of standard practice. Our second game, Sleepy Town, generated even greater variability, which was again significantly different from both. Differences between these distributions were validated using seventeen global, relative, and coverage measures. Our findings motivate the need for researchers and designers to move away from standard practice, and for the community to develop new ecologically valid data collection protocols. We believe that FTL is a good first step solution, as FTL requires little effort to implement, being built on tools already present in standard data collection applications, and that elicits greater variability.

CHAPTER 13: DISCUSSION AND FUTURE WORK

Like the primordial Buddha beneath the Bodhi
My pseudo-mind pseudo-wandered
I climbed and I clambered
And I ambled upon some understanding

King Gizzard & The Lizard Wizard
Han-Tymi, The Confused Cyborg

Although we have demonstrated significant advances in continuous custom gesture recognition, The Dollar General is still an imperfect solution, ripe with possibility and potential. In this chapter, we discuss lessons learned as well as highlight areas that require further investigation, including those that extend beyond topics covered in earlier chapters.

13.1 On the Use of Direction Vectors to Represent Human Motion

We found through the development of several techniques that direction vectors afford a powerful and intuitive representation of human motion. Our initial effort with Penny Pincher [238] demonstrated that their compact form and amenability to variation leads to efficient computation and robust recognition. Direction vectors are both scale and position invariant, and when we relate corresponding vectors between trajectories we can determine whether two samples move identically though space. The performance boost we observe when we compare trajectories through direcction vectors is because a vector-based similarity measure can leverage dot products, requiring only addition and multiplication while avoiding library calls to more expensive functions.

By adding elasticity (Jackknife), we further improve our ability to match patterns across numerous modalities, including but not limited to accelerometer, skeleton, pen, touch, mouse, and sound

input. Elasticity allows for temporal variation where trajectory subsequences may be longer or shorter then that which we compare against. This idea relates to gesture path stochastic resampling in that we shrink and stretch subsequences to create variations of a given gesture sample. In other words, since one can represent motion as a set of vectors, vector length changes naturally alter a trajectory's shape.

We continued to exploit direction vectors with Machete by employing continuous dynamic programming. Since position and scale are unknown in advance, a vector representation allows us to align incoming data based on shape, which is challenging with only position data. However, despite the numerous advantages afforded by direction vectors, there are also some shortcomings we have learned about that are worth highlighting:

1. *Unknown position*: Subtle changes in position can be lost over long sequences such that accumulated drift may have little impact on a vector-based dissimilarity measure. These differences may be easier for a position based measure to detect.

2. *Rotation dependence*: Motion vectors are position and scale invariant, but not rotation invariant. This is because we work with raw data where possible, and without a priori knowledge of the underlying data, we cannot assume what types of rotations are valid, if any at all. For example, Jackknife is able to recognize gestures produced in a sound field by detecting frequency shifts in binned spectrum data. However, sound wave data rotations have no practical meaning, yet skeleton data rotations do have meaning. One can overcome this limitation at the application layer by training a recognizer with manually rotated samples or by asking users for rotated samples. Another option is to monitor pose information and align input data with a canonical pose before passing input to a recognizer. Regardless, rotation invariance support with direction vectors may require a fresh approach.

3. *Sensitivity to resampling rate*: Recognizers that use Euclidean distance to measure dissimi-

larities between query and template points generally improve as the resampling rate grows. However, despite our best efforts to filter input device signals, the angular difference between corresponding direction vectors may degrade as the resampling rate increases because of noise. Similarly, direction vector orientation on corners and cusps will vary depending on where their end points land on a trajectory. This variance may result in a misrepresentation of direction and false dissimilarities between otherwise similar patterns. A better representation of direction that considers local and global properties near vector end points may be able to overcome these issues.

4. *Static Poses*: We have found that direction vectors are good for representing motion, especially for representing the shape of motion through a multidimensional space. However, a significant portion of gestures are static poses that cannot be well represented by constructs that describe motions. At best, one may include time as a input sample component (*e.g.*, x, y, z, t) to create a dynamic element that our methods can use to detect static patterns. If there is motion going into and out of a pose that training samples capture, then we can potentially detect static gestures; otherwise, we are still unable to differentiate specific poses, only rest periods.

Based on these findings, we believe it will be useful to investigate data representations that combine position and direction information. How this representation will manifest itself in practice is unclear, but should be considered as part of our future work.

13.2 On the Possibility of Establishing a Synthetic Data Generation Continuum

We found in Chapter 8 that data augmentation can improve recognizer accuracy regardless of whether one uses a parametric or non-parameter recognizer, and we provided evidence that GPSR

is an especially adept synthetic data generation technique. In Chapter 9, we introduced splicing and Mincer, both of which generate negative data in a way that serves unique purposes. By combining Mincer with GPSR, we were able to reliably estimate suitable rejection thresholds.

In developing these techniques, we learned there are unique perspectives on synthesis and how one should go about generating synthetic data. Our initial GPSR implementation tried to replicate population statistics by minimizing shape error [251] in order to train recognizers with synthesized data. On the other hand, VKM was concerned with score distributions (not realism), wherein synthetic samples were measured for their similarity to other training samples, but were not included in the training set. We wanted to ensure measurements in aggregate formed a representative distribution. Consequently, malformed samples were not a problem because they did not become templates. From this difference, we developed two different optimal-N equations that solved these two different problems. However, there is a third perspective to consider. We sometimes care about subjective realism, where we want to generate synthetic samples that appear genuine to human observers, but are indistinguishable from other real samples. Each of these approaches may result in distinct data distributions.

We can look at negative synthetic data through the same lenses, but add yet a forth perspective. There are situations where we wish to condition the distribution based on a constraint. For VKM, we were required to generate synthetic data near class boundaries, rather than uniformly through the entire sample space as we do with splicing. This difference not only impacts the distribution but also the gesture shapes themselves. When combined with the above, it will be interesting to see if we can characterize the relationship between these operational modes and determine if they can be linked through a mathematical relationship. Is there a continuum that underlies these modes? Can GPSR and Mincer be combined into a single algorithm with a single knob that ranges from ultra realistic to completely random? What are the implications of such a continuum on the synthesis process itself? These are the kind of questions we believe require further attention.

13.3 On the Advancement of Correction Factors

As discussed throughout this work, researchers have proposed a number of nearest neighbor recognizers that vary in how they measure the dissimilarity between query and template samples. Some variants cater to 2D gesture recognition [8, 9, 143, 248, 250, 267], while others generalize to 3D [128, 129, 235], but all methods treat gestures as a set of points or vectors that must be aligned, matched, and measured. Jackknife is one such approach that, like its contemporaries, uses a pairwise scheme to measure differences between spatially resampled trajectories, but unlike its contemporaries, also uses *correction factors* to augment and improve the final measure. As a reminder, correction factors use descriptive statistics that measure information embedded in a distribution of points, where the statistic of two samples are combined into an dissimilarity measure that augment (inflate) the underlying score.

Correction factors afford several advantages. They capture information that traditional nearest neighbor approaches may miss, such as density. They are so far generally straightforward—easy to understand and implement. They are recognizer agnostic and can be combined with other dissimilarity measures. Similarly, they can be chained together into a series of corrections and have potential to significantly improve recognition accuracy [15, 233, 235]. For these reasons, we believe that correction factors deserve more attention.

We believe well-formed correction factors will possess several desirable properties as outlined below:

1. Computationally efficient to allow for continuous, online processing.
2. Straightforward, using intuitive notions so as to facilitate understanding, implementation, and debugging.

3. Robust against variation such that inflation is minimal between within class pairs and large between different classes. In a customization context, we may be unable to estimate the statistic's distribution.

4. Independent in aggregate. New correction factors should work to measure information not captured by other factors to minimize correlations.

The proposed criteria will feel familiar as they align with the goals of recognizer design in general. However, one major difference between feature based gesture recognition and correction factors is that the former learns distributions from data that a recognizer uses to estimate a gesture's position within a feature space relative to its class models. Correction factors do not have advanced knowledge of the distributions and cannot, for example, adjust values based on variance, which is a challenge correction factors must overcome.

13.4 On Continuous Online Training with Application Query Data

We understand from this and other $-family related research that recognizer performance generally improves with training set size. However, it is unclear in what way performance will change if we train a recognizer with both query and application data. This is an important question because significantly more query samples are realized than are training samples. In other words, a researcher engaged in prototyping will necessarily and repeatedly use their software in order to test, debug, and evaluate its efficacy; and he or she will also ask peers to do the same. In this way, a system will receive numerous queries that the system can use to continuously retrain the recognizer. It will be useful if we can in fact leverage this data.

One obstacle is that recognizers are imperfect, yet to retrain a recognizer with query data requires that we trust its initial classification decision. A second obstacle is bias. If we assume that users

invoke functions non-uniformly, then training set size bias may impact performance. Further, new functions that arrive in later software iterations will similarly have less history and training data. Based on these concerns, we ask more specifically: Will $-family recognizer accuracy improve over time if they continuously retrained with imprecise and biased application data? If custom gesture recognizers can leverage application data in combination with prototype selection (Chapter 11), it will further motivate their use.

13.5 On the Possibility of Customizing Accuracy Using Information Theory

A fundamental question we do not address in this work but that impacts user experience follows: How accurate does a custom gesture recognizer have to be to motivate its use in an application? Although subjective experience is important, we can also look at this problem from a purely quantitative perspective and ask how accurate does a recognizer have to be to outperform alternative input interaction techniques, such those afforded by WIMP-based user interfaces. If we cast this problem into an information theoretic framework, then it may be possible to answer such questions. In this framework, we treat each command as a communication and our goal is to maximize throughout.

Let us say that an application implements a finite set of commands $\mathbb{C} = \{c_i \mid i = 1 \ldots N\}$ that are accessible via a user interface, regardless of the input device type. Further, each command has a certain probability $p(c_i)$ of being invoked by a user. This probability distribution may vary by use case and user, but for our discussion, let us assume uses and users are sufficiently similar such that a common distribution provides an adequate representation. The amount of information that a command message communicates is its information content $-log_2(p(c_i))$ measured in bits. To derive throughout from information content and be able to compare different interaction techniques, we simply divide by the technique's expected command production time. Noting that users make

mistakes, input devices are noisy, and software is imperfect, we must also consider error recovery time. Putting these ideas together, we define *expected* throughput TP for a given application implementing command set $\mathbb{C}$ and interaction technique I as follows:

$$TP = \sum_{i=1}^{N} \frac{-p(c_i)\log_2 p(c_i)}{\left[\frac{1}{A_I(c_i)} - 1\right](R_I(c_i) + T_I(c_i)) + T_I(c_i)}, \tag{13.1}$$

where accuracy $A_I(c_i)$ is the probability that command c_i is correctly input and recognized, $R_I(c_i)$ is the expected time it takes to recover from an error when command c_i was intended by the user but the system detects a different command, and $T_I(c_i)$ is the expected command production time. The bracketed factor is the negative binomial distribution mean, expressing the expected number of failures one will encounter before the correct command is input. One can further break production time down into recall and execution components, though the combined value is sufficient for our discussion.

As part of our future work, we propose to formalize throughput in a gesture recognition context so that one can directly compare how well their recognizer works compared to other interaction techniques. One can similarly use throughput to understand how well a gesture set will work with respect to a specific command set based on probable command use. Using this approach, there may also be an opportunity for designers to make informed decisions based on throughput.

For arbitrarily high accuracy, we see that the left summand of the denominator tends toward zero, and throughput is simply the time it takes to produce a command. Yet, a user may accidentally click on the wrong menu item, or a recognizer may missclasify a gesture. In this case, he or she will correct the error and retry the command, which is to say that both production time and accuracy are highly relevant in evaluating an input interaction strategy. To maximize throughout, a designer must minimize error and production time.

Well crafted gestures already have generally low production times (1—2 seconds), and the command probability distribution is likely fixed. This leaves us with only accuracy that we may manipulate. Here we propose adjusting recognizer sensitivity as a function of command probability. For example, with highly probable commands, one can lower the rejection threshold, whereas for less probable commands, one may increase the rejection threshold. With sufficient training data through online data collection or synthesis, one can further estimate throughout for varying thresholds to select that which maximizes performance. For this reason, future work should include an exploration of how one can tie together throughout and threshold selection.

13.6 On Using Context to Improve Recognition Accuracy

A gesture can be understood in context given that one's actions depend on the state of affairs in which an action is taken. Researchers have worked to exploit context across a variety of applications [20, 53, 73, 90, 107, 124, 125, 135, 138, 165, 166, 209, 269]. In our own work [236], we explored three unique ways one can use context to improve gesture recognition: continuous online retraining using distributions conditioned on context, discarding out-of-context gestures to accelerate computation, and conditioning recognizer score distributions based on context. To better understand how we can use context with customization, let us first define the gesture prior for a game-based virtual environment, which requires player and environmental context.

13.6.1 Player Context

We define player context, Φ , as an ordered set of state information that updates once per frame. An illustrative example follows:

$$\Phi = (\phi_p, \phi_l, \phi_r, \boldsymbol{\phi_o}, \boldsymbol{\phi_f}), \tag{13.2}$$

where ϕ_p is the player's primary interaction state, $\phi_p \in \{p_0, p_1, p_2, ...\}$. Typically, player context is equivalent to state information defined in a game. For example, one state may be associated with free fall (p_0) when a player drops from a ledge or climbing (p_1) when the player grips a wall. ϕ_l and ϕ_r are the left hand and right hand substates with $\phi_l \in \{l_0, l_1, l_2, ...\}$ and $\phi_r \in \{r_0, r_1, r_2, ...\}$. Each l_x is an enumerated left hand substate and each r_x is similarly an enumerated right hand substate. For example, r_0 may indicate that the right hand is locked onto a crevice when $\phi_p = p_1$. A three-dimensional Cartesian coordinate, $\boldsymbol{\phi_o} =< x, y, z >$ is used to represent the player's position and the Euclidean vector $\boldsymbol{\phi_f} =< x, y, z >$ is the player's forward direction (with zero rotation about the forward axis). With the above definition, we have sufficient information to track players through an environment for a variety of games.

13.6.2 Environmental Context

Environmental context is the set of all virtual objects with which a player may directly interact, defined as

$$\Psi = \{\psi^0, \psi^1, \psi^2, ...\}, \tag{13.3}$$

where each ψ^x is an individual virtual object in the VE and x is an enumerated identifier. A typical virtual game environment is dynamic and ever changing, therefore, Ψ is constantly morphing as objects are created, destroyed, and manipulated. Example objects include static scene geometry, enemies, and artifacts. Like player context, each virtual object is defined as an ordered set of states and attributes:

$$\psi^x = (\psi_i^x, \psi_o^x, (\psi_{s0}^x, \psi_{s1}^x, ...)). \tag{13.4}$$

The virtual object type identifier is defined as $\psi_i^x \in \{i_0, i_1, i_2, ...\}$ where each i_x is an enumeration of all possible object type identifiers. Further, each object's position is given by the Cartesian coordinate $\psi_o^x =< x, y, z >$. State information specific to the object type is captured in the sequence

$(\psi^{x}_{sy}, ...)$. This definition can be easily extended to include other game relevant information, including health, ability, strength, endurance, and so forth.

13.6.3 Gesture Prior

Each gesture $g_i \in \mathscr{G}$, is assigned a per frame probability based on current context by application logic, which yields the gesture prior:

$$P(g \mid \Phi, \Psi) = \frac{F_g(\Phi, \Psi)}{\sum_{i \in \mathscr{G}} F_i(\Phi, \Psi)}, \tag{13.5}$$

with $F_g(\Phi, \Psi)$ defined as:

$$F_g : \Phi \times \Psi \rightarrow \Re_0^+, \tag{13.6}$$

where F_g is a real valued function. One special case of this is an indicator function that maps a gesture prior to zero or one.

There are at least two related ways in which one can use gesture priors to improve recognition accuracy. First, the prior can be used to augment Equation 13.1 gesture probabilities, thereby improving average throughput when appropriate actions are taken to adjust sensitivity from context. Second, one can use gesture priors to adjust rejection thresholds by manipulating F_β to favor precision or recall, or it may be possible to apply rejection threshold scaling as we do with inflation factors and training set sizes (Chapter 9). Since customization relies on limited training data, we see context as a useful and currently underutilized tool we can use to improve performance.

13.7 On the Straightforwardness of The Dollar General

Throughout this work we emphasized straightforwardness where possible. Our motivation came from the virtues outlined in Chapter 2.7 that in summary offer a degree of freedom. Whether one's inexperience or ineptitude constrains their ability to effectuate a gesture-based user interface design, we believe that the techniques presented herein help one overcome their limits. The effects of enforcing a $-family inspired design philosophy have, however, also had several positive effects that extend beyond its intended purpose:

- We believe that by adhering to this design philosophy, some methods turned out better than they would have otherwise. For example, our initial version of Machete was notably more complex than it is now, wherein we normalized column path lengths in a way that its mathematics cumbersome. In a reevaluation of our design, we unlocked a simpler approach that also happen to work better. If we were not driven by a desire to keep things as simple as possible, we may have missed this opportunity.

- As with constrained writing, where the writer is forced to abandon their usual constructs, phrasing, and style, we must on occasion abandon our usual algorithmic approaches. Once constrained, one is forced to consider alternative approaches that would have otherwise eluded them. This constriction sometimes leads to insightful solutions, new constructions, and works of art, such as the French novel Le *Train de Nulle Part* by Michel Dansel that despite having no verbs, comprises surprisingly understandable passages. In avoiding complexity specifically, we stumbled on a resampling twist that enabled GPSR, we were better able to exploit the power of direction vectors, and pursued a simple copy-paste idea that became Mincer; and moreover, these advancements all serve customization, achieving state-of-the-art recognition accuracy. Although, one may work under any sort of constraint and perhaps see certain benefits, adherence to $-family principles in our view has a higher pur-

pose that serves the community. For this reason, we believe one should always keep these principles in mind, regardless of their pursuit.

13.7.1 Is The Dollar General Practical

We acknowledge that not all Dollar General components meet the $-family design requirements. Perhaps the worst offender is Pitch Pipe which uses Fourier transforms to estimate noise and low frequency power amplitudes in order to find an appropriate cutoff frequency for a given input signal. The mathematics are simple, involving only summations and trigonometric functions, but Fourier analysis itself takes considerable time to understand and may be difficult to debug when uncertainty in an implementation surfaces. In this regard, Pitch Pipe is not $-family compatible, but we believe it is still practical given that an advanced undergraduate in computer science will likely have been exposed to enough ideas they can understand our design. What about the remaining techniques, however?

Let us revisit the design philosophy's tenets with a critical eye toward TDG:

1. *Independence*: No proposed method requires external library support to solve a complex problem, *e.g.*, for optimization, integration, or random sampling, since all techniques are based on straightforward mathematics. We further provide pseudocode for all methods.

2. *Foundational Mathematics*: All mathematical constructs herein involve only basic scalar and vector arithmetic as well as simple statistics, namely uniform random sampling. We do not use calculus, advanced linear algebra, or differential equations. Since vectors are a widely used geometric construct and the inner product is a trivial calculation, we do not consider our use of vectors to be advanced.

3. *Relatable Representations*: The Dollar General heavily relies on direction vectors to represent human motion. This representation is commonly used throughout the sciences, and many students will have already been exposed to vectors in their primary education, even before attending college. We believe the idea that the difference between two points is a vector describing their displacement will likely have been reinforced many times over by the time one is ready to engage is user interface design engineering or research.

4. *Basic Algorithms*: Most algorithms we use are fundamental, involving only trivial list iterations. One exception is that we use dynamic time warping in Jackknife and continuous dynamic programming in Machete. Both of these, as implied by their names, involve dynamic programming, which is considered by some to be a slightly more advanced topic. In this sense, one can argue that we do not strictly adhere to the basic algorithms rule. However, with a proper explanation and well documented pseudocode, we believe understanding DTW and CDP is tenable.

5. *Competitive Accuracy*: We have shown through numerous evaluations that TDG is able to achieve high accuracy that is competitive with alternative techniques.

6. *Customizability*: Similarly, though the same evaluations, we have shown that TDG performs very well with little training data.

By examining the design criteria and reflecting on our approach, we feel confident in claiming that TDG is practical, but that only some parts are fully $-family compatible. Namely, Penny Pincher, GPSR, and Mincer. We believe that Jackknife and Machete are still very accessible, but will require additional effort for some individuals to fully grasp; and many can still use these techniques to rapidly prototype user interfaces.

Perhaps a larger issue is the amount of coding required to implement the full TDG pipeline. We

essentially solve four related but unique problems: filter parameter selection, segmentation, classification, and rejection. Each part has several components of which only some overlap. Even if no parts were overly complex, a significant engineering effort is required to bring each component online. In this regard, it may be useful to look at how we might bring these systems together. For example, can segmentation and classification be unified into a single module using principles and ideas discussed throughout this work?

13.7.2 Personal Reflections

Speaking now as the primary author of this work, I want to highlight some of my experiences in working with $-family recognizers in general, and TDG specifically. Although I am biased by my relationship with this work and personal experiences do not generalize, hopefully these thoughts will help color the discussion. Throughout my career, I have implemented a variety of algorithms including those for Reed-Solomon error-correcting codes for redundant storage systems, cache coherency mechanisms for IO controllers, distributed data layout designs, face recognition with AdaBoost, global illumination via ray tracing using data acceleration structures on GPU, and 2D grammars for sketch-based mathematics, among others. I implemented several of these while working at nStor and Xyratex as an undergraduate, several in the years before graduate school, and of course several more during graduate school. Reflecting on these experiences, I can confidently claim that $-family recognizers were among the easiest class of machine learning techniques to understand and implement for gesture recognition, and among the most straightforward of all mentioned algorithms in general. In my view, many TDG components are equally convenient, including GPSR and Mincer, as well as Jackknife's predecessor Penny Pincher. DTW did take extra effort to fully appreciate, in part because my initial introduction to this measure was lackluster; though I never had any problems implementing, debugging, and successfully exercising DTW, which is why I brought DTW into Jackknife. Despite this minor disparity, Jackknife and Machete

(once formulated) have been easier to understand, execute, and modify than most other projects mentioned above. (Even to this day I still do not fully understand Reed-Solomon codes!) I hope that others feel the same, and that the art continues to advance.

13.8 Conclusion

In this chapter, we presented lessons learned and future directions. We found that although we have made significant process toward providing the community with a continuous custom gesture recognizer, there are still several exciting areas for inquiry remaining. Some areas involve direct pipeline modifications that improve measurement or straightforwardness, whereas others are areas that extend the pipeline, such as to include context into the rejection threshold calculation. We hope that students, practitioners, and experts alike find value in TDG, and we look forward to the future of this technology.

CHAPTER 14: CONCLUSION

Tick tick tick, do you recognize the sounds,
As the grains count down,
Trickle down right in front of you?

Perfect Circle
Hourglass

Being fundamental to interpersonal communication, gestures have also become an important part of the human computer interaction experience. Gestures are natural. Gestures communicate. An individual expresses their self through physical metaphor, often unconsciously and without mindfulness to enhance discussions and express ideas. To better facilitate this kind of interaction at the user interface boundary, we require recognition techniques that support customization, yield high accuracy, and are straightforward, so that design and progress are not stifled by lack of tools, knowledge, and techniques. Our work aimed to address this problem.

We have presented a continuous custom gesture recognition pipeline that is device agnostic, generally accessible, accurate, and computationally efficient. As much as possible we have adhered to the following design guidelines: any technique should work with limited training data so as to enable customization; each part of the pipeline should use only basic mathematics, intuitive data representations, and basic algorithms so that the combination thereof can be called straightforward; and one can implement the entire system without requiring external tools support. The breadth of problems we tackled in our pipeline is undoubtedly large. We provide solutions for low-pass filter parameter selection, segmentation, recognition, synthesis, rejection, prototype selection, and data collection. Although continuous gesture recognition is holistically a major undertaking, we believe that The Dollar General significantly advances the state of the art.

APPENDIX A: SPLICING EVALUATION ON LOW-ACTIVITY DATA[1]

[1]This appendix contains previously published material adapted from the following article: Taranta II, Eugene M., *et al.* "Jackknife: A reliable recognizer with few samples and many modalities." Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. 2017. See Appendix C for associated copyright information.

In this appendix, we report on an initial version of our automatic rejection threshold technique based on GPSR and splicing. An improved version of this system is discussed in Chapter 9. We start with an overview of the problem and solution.

How one can determine an appropriate per template rejection threshold from only a minimum amount of training data remains a difficult problem. With sufficient amounts of training data, a recognizer can estimate within class score probability densities and select thresholds sufficiently low enough to prevent type II (false negative) errors. One can also use negative samples that are non-gesture sequences to help control type I (false positive) errors by ensuring that a rejection threshold is sufficiently high enough to prevent false positives. With access to both positive and negative samples, one can select a threshold that minimizes both error types, which is the strategy we adopt here and carry over to VKM (Chapter 9). In this work, we assume that only a minimum number of positive training samples are given, as little as one or two per gesture class. This limitation implies we need to somehow synthesize both negative and positive samples.

Herein, to create negative samples, training samples are spliced together to create semi-nonsense, noise-like sequences. We favor this approach because we desire that negative samples have parts of real gestures embedded within to ensure that Jackknife can reject sequences that partially resemble but are not actually real gestures. To generate synthetic negative samples, we randomly sample k training samples and from each sample we randomly sample $(n-1)/k$ sequential direction vectors. These direction vectors are concatenated together to form a negative sample. We then compare each template with the negative sample using DTW and save the results per template. This process is repeated a number of times, after which the scores are z-score normalized per template.

The generation of synthetic positive samples requires a different approach. Gesture path stochastic resampling (GPSR) [232] is a new synthetic data generation technique developed specifically for 2D gestures and rapid prototyping. A gesture path is non-uniformly resampled to $n+x$ points, after

which the distance between subsequent points is normalized to unit length, and finally x random points are removed. GPSR was shown to produce realistic results for pen and touch gestures; however, for our use, we do not require realism. Rather, we only need to be able to create a distribution of samples that when evaluated with DTW generates a score distribution similar to the true distribution, and we found GPSR works well for this purpose. For additional information on implementation details and pseudocode, we refer the reader to [232]. We use GPSR to create synthetic positive samples that are scored with DTW against their seed samples. These within class scores are then z-score normalized using the mean and standard deviations generated from the negative samples evaluation above.

Now we are able to determine a rejection threshold. With the distribution of positive and negatives samples (all of which have been z-score normalized), a standard deviation λ is selected that minimizes the aggregate false and negative positive count. Since we trained with a noise-like pattern, our goal is to be able to reject idle-like motion[2]. The per template rejection threshold δ is then:

$$\delta_i = \mu_i - \lambda \sigma_i, \tag{A.1}$$

where μ_i is the mean of the negative sample scores relative to each template i, and σ is its standard deviation. As we will show in an evaluation, this approach appears to select thresholds near the optimum for low-activity data.

A.1 Examination of the Distributions

To examine the distribution of scores generated by Jackknife for a specific dataset, we do the following: For a given subject, a recognizer is trained with one random sample per gesture class.

[2]To increase tolerance to higher energy patterns, the minimization can non-uniformly weight the importance of true and false positives.

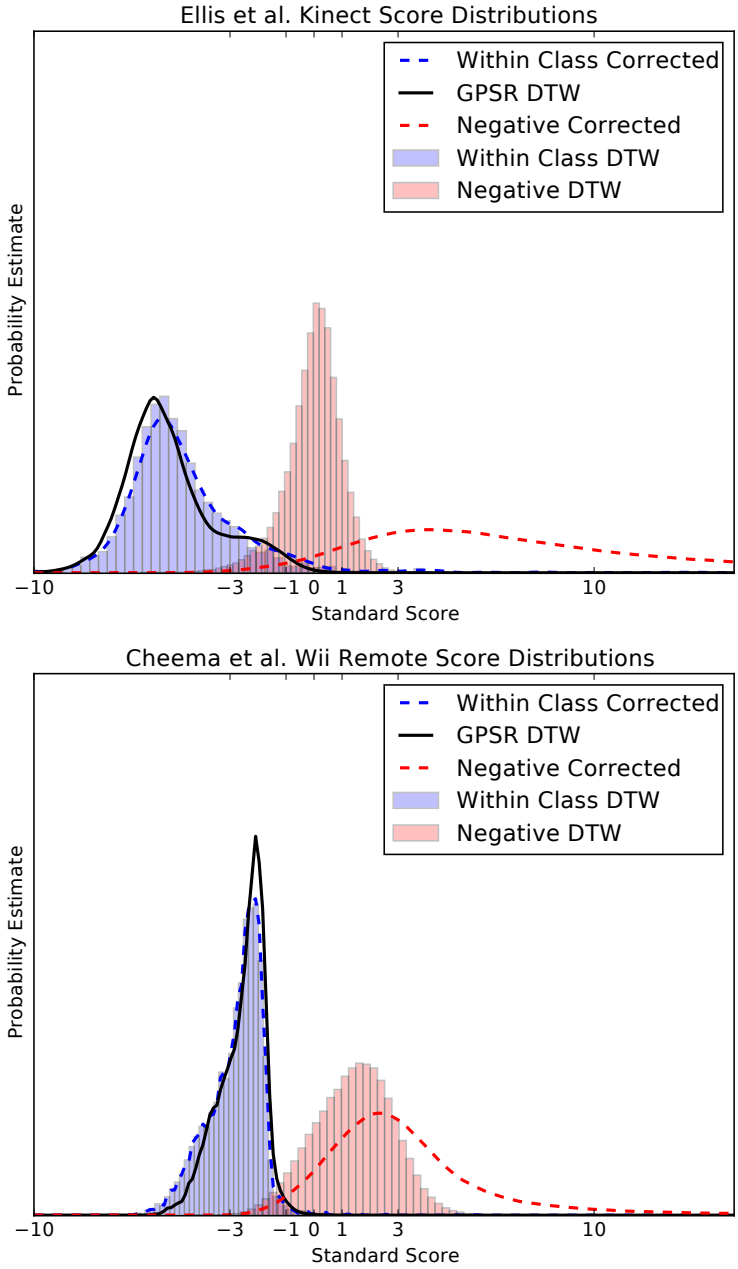


Figure A.1: Normalized distributions of within class and negative samples before and after correction factors are applied, as well as the synthetic positive sample distribution for Ellis *et al.*'s [63] and Cheema *et al.*'s [38] datasets.

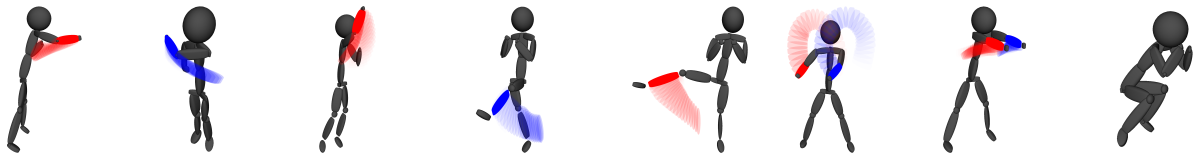
All z-score normalized synthetic scores generated as part of the training process are saved for analysis. An additional, but unique test sample per gesture class is subsequently evaluated, and the within class DTW score of each test sample, before and after correction factor inflation, are similarly z-score normalized and saved. Last, new synthetic negative samples are again generated, scored, and saved. This process is repeated 10 times per subject and all results are combined into a single set of distributions.

Results are shown in Figure A.1, and we note a number of observations. First, the within class samples distribution is well separated from the negative samples, having only a small amount of overlap, which we perceive as a desirable property for a noise pattern; a poor noise distribution would be far away from the within class samples and result in unreliable rejections thresholds (no false positives or false negatives). Second, the within class corrected DTW scores are remarkably close to the true distribution of the uncorrected DTW scores, whereas the distribution of negative samples are shifted right, away from the within class distribution. This observation suggests that the correction factors are doing their job, although further analysis in the next section will quantitatively confirm this. Finally, the positive samples generated using GPSR form a score distribution that is quite near the true distribution. This does not imply that the synthetic samples are realistic, but it does help build confidence that GPSR can be used to help find a reasonable rejection threshold when used in combination with synthetic negative samples.

A.2 Evaluation of Continuous Data

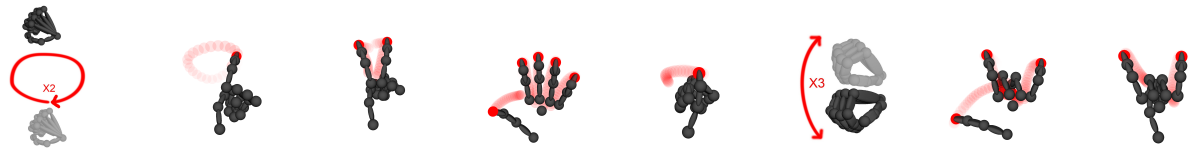
To evaluate the effectiveness of our approach in rejecting non-gesture sequences from a continuous data stream, we collected test data from a pool of 40 students (30 male and 10 female) at the University of Central Florida, ranging in age from 18 to 28. The participants were divided into two groups where the first group worked with a Kinect and the second group worked with a Leap

Table A.1: List of 14 Kinect gestures used in our contiguous data study. Note that L/R indicates there is a left and right side version of the gesture.



Jab L/R | Hook L/R | Uppercut L/R | Kick L/R | Sidekick L/R | Cartwheel L/R | Push | Duck

Table A.2: List of 8 Leap Motion gestures used in our contiguous data study. Each gestures starts with one's hand in a first and ends in the same position.



Fist Circles | Index Circles | Snip Snip | Explode | Sideways | Knock | Love | Devil Horns

Motion. We collected segmented training data of the gestures shown in Tables A.1 and A.2 as well as a continuous, uninterrupted session of the sample gestures performed in random order with repetitions, which is discussed in detail below. Each participant took around 15 to 20 minutes to complete all tasks, including a pre-questionnaire to collect demographic information, and all were compensated $10 for their time.

The experimental setup comprised a 50 inch Sony Bravia HDTV and a Microsoft Kinect 2.0 or Leap Motion. The Kinect sensor was mounted above the HDTV using a mounting device and was kept stationary throughout all sessions. The Leap Motion sensor was mounted on a desktop in front of the television and was kept in place through a single session using tape, except in one case where the participant was left handed and the sensor's orientation was changed to accommodate for their comfort. Further, a box was placed between the participant and device so one could rest their arm and avoid fatigue, which also helped to control the distance and orientation of a participant's hand during the study.

Table A.3: T=2 Percentage accuracy results for various rejection thresholds on the Kinect gesture dataset shown in Table A.1. The last entry is the threshold that was automatically selected.

λ	F_1 Score	Precision	Recall	FPR	Pruned
	$\mu(\sigma)$	$\mu(\sigma)$	$\mu(\sigma)$	$\mu(\sigma)$	$\mu(\sigma)$
1.5	95.7 (5)	95.4 (5)	96.1 (6)	0.3 (0)	89.2 (6)
1.75	96.0 (6)	96.0 (6)	96.1 (7)	0.3 (0)	90.1 (5)
2.0	96.9 (5)	97.3 (5)	96.5 (6)	0.2 (0)	90.9 (5)
2.25	96.4 (6)	97.5 (4)	95.5 (7)	0.2 (0)	91.7 (5)
2.5	95.9 (6)	97.5 (4)	94.5 (8)	0.2 (0)	92.6 (4)
-2.75	95.1 (5)	97.5 (3)	93.0 (8)	0.2 (0)	93.3 (4)
1.98	96.4 (5)	96.8 (5)	96.1 (6)	0.2 (0)	90.8 (5)

Table A.4: T=2 Percentage accuracy results for various rejection thresholds on the Leap Motion gesture dataset shown in Table A.2. The last entry is the threshold that was automatically selected.

λ	F_1 Score	Precision	Recall	FPR	Pruned
	$\mu(\sigma)$	$\mu(\sigma)$	$\mu(\sigma)$	$\mu(\sigma)$	$\mu(\sigma)$
1.5	93.8 (5)	93.3 (7)	95.0 (7)	0.8 (0)	54.8 (7)
1.75	94.4 (5)	95.0 (5)	94.4 (8)	0.6 (0)	58.1 (7)
2.0	94.3 (5)	95.8 (5)	93.3 (8)	0.5 (0)	62.5 (7)
2.25	94.0 (7)	97.7 (3)	91.5 (11)	0.3 (0)	66.2 (6)
2.5	92.8 (8)	98.1 (3)	89.4 (13)	0.2 (0)	70.2 (6)
-2.75	90.9 (10)	98.3 (2)	86.3 (15)	0.2 (0)	73.7 (5)
1.95	93.8 (6)	95.7 (5)	92.5 (9)	0.5 (0)	61.8 (8)

A.2.1 Data Collection

We developed a data collection utility with two modes using Unity 3D v5.4. The first mode allowed us to collect clean, segmented samples of each gesture for training. A participant was asked to perform a specific gesture that was initiated by a count down. The sequence was then resampled

and replayed to ensure correctness. If an error occurred or the participant was unhappy with the replay, the sample was discard and recollected. This process was repeated twice so two training samples per gesture were collected. Note that all gestures were demonstrated to a participant before any data was collected. The second mode enabled us to collect a continuous stream of gesture articulations. Our application prompted the participant to perform a specific gesture selected at random, and once we confirmed the correct action was performed, we manually advanced the application to the next gesture. Otherwise, if an error occurred, *e.g.*, a malformed gesture or loss of tracking, the action was retried. A proper example of each gesture was collected three times so that there were a total of 14×3 valid Kinect gestures and 8×3 valid Leap Motion gestures included in the *unsegmented* stream.

We noticed the Kinect lost track of some participants more frequently than others, which may be related to the texture and colors of the participants clothing. In such cases the skeleton appeared jittery on the screen and gesture were repeated as necessary.

The Leap Motion device also frequently lost tracking and detected incorrect hand poses, such as reporting that both the index and middle fingers were extended when, in fact, only the index was so. These problems were exacerbated with 5 participants who had smaller hands. Also, we actually collected data for 9 Leap Motion gestures, but found one gesture (not shown in Table A.2) was difficult for participants to perform. This gesture was removed from our analysis.

A.2.2 Results

Using data collected from each participant, we trained Jackknife and replayed the participant's session. All classification results including true and false positives (tp and fp) as well as true and false negatives (fn and fn) were averaged into an overall result. Additional parameters were tuned as follows: the recognizer was run once every 10 frames using the last four seconds of

buffered frame data[3]; once an action was detected, the buffer was cleared and the recognizer was suspended for 2 seconds, which we believe is sufficient time to prepare for the next gesture; and a gesture was considered as executed if Jackknife returned the same result twice in a row. Based on our experiences, these are fairly reasonable criteria, which can be tuned to match a practitioner's specific requirements.

We reran the above procedure several times for different levels of λ used to set the rejection threshold (see Equation A.1). Table A.3 shows results for our Kinect continuous data test, and Table A.4 shows results for the Leap Motion test. The rejection threshold is important in balancing precision ($tp/(tp+fp)$) and recall ($tp/(tp+fn)$), and the F_1 score is the harmonic mean of these measures. Our goal is to therefore maximum F_1. In the Kinect test, we are able to achieve 96.9% at $\lambda = 2.0$, which is close to the automatically selected threshold $\lambda = 1.98$. Similarly, the maximum F_1 score of 94.4% for the Leap Motion test occurred at $\lambda = 1.75$, which again is near the automatically selected $\lambda = 1.95$. Recall that the automatic threshold is selected so that the false and true negatives generated from synthetic data is minimized, which appear to be appropriate for idle movement in between actions. However, λ can also be increased to allow for a larger variety of non-gesture actions to be performed at the expense increased false negatives. According to our results, the affect on the F_1 score is not too large, where the improvement in precision may well be worth the loss in recall for many applications. These results suggest that Jackknife with splicing is useful for working with continuous data.

[3]We used four seconds because some gestures were performed slowly by some participants; though we could have used a shorter duration in most cases. Since the gesture paths are resampled to $n = 16$, idle frames do not significantly contribute to the shape of the action.

A.3 Conclusion

Our evaluation reveled that rejection threshold selection using synthetic data generation with GPSR and splicing works well on low-activity data given that we were able to achieve high accuracy. The Voight-Kampff Machine discussed in Chapter 9 builds and improves on these ideas.

APPENDIX B: PSEUDOCODE

B.1 Common Methods

Algorithm B.1: DISTANCE (POINT a, POINT b)

```
/*** Get number of components per point.                                  ***/
/*** Note that the dimensionality of a and b must be equal.               ***/
m ← |a|

/*** Calculate Euclidean distance between the two points.                 ***/
d ← 0
for (i ← 0; i < m; i ← i + 1) do
    d ← d + (a[i] − b[i])^2

return √d
```

Algorithm B.2: PATH-LENGTH (POINTS *points*)

```
d ← 0
for i ← 1 to |points| − 1 do
    d ← d + DISTANCE(points[i − 1], points[i])
return d
```

Algorithm B.3: BOUNDING-BOX (POINTS *points*)

```
/*** Get number of components per point, i.e., point dimensionality.      ***/
m ← |points[0]|

/*** Set min and max to the first point.                                  ***/
minimum ← points[0]
maximum ← points[0]

/*** Extract component-wise min/max over all points.                      ***/
for (i ← 1; i < |points|; i ← i + 1) do
    for (j ← 0; j < m; j ← j + 1) do
        minimum[j] ← MIN(minimum[j], points[i][j])
        maximum[j] ← MAX(maximum[j], points[i][j])

return (minimum, maximum)
```

Algorithm B.4: DIAGONAL (POINTS *points*)

```
minimum, maximum ← BOUNDING-BOX (points)

return DISTANCE(minimum, maximum)
```

B.2 Jackknife

Algorithm B.5: JACKKNIFE-CLASSIFY (*query*, *templates*, *resample_rate*)

```
/*** ...                                                                    ***/
query ←JACKKNIFE-PREPARE(query, resample_rate)
/*** Go through each template and see which best matches the query.         ***/
best_score ← ∞
gesture_class ← none

for (template in templates ) do

    /*** Get base inner product DTW score and inflate with corrections.     ***/
    score ← DTW-IP (query.vectors, template.vectors, ratio = 0.10)
    score ← score / DOT-PRODUCT(query.extents, template.extents)
    score ← score / DOT-PRODUCT(query.abs_distance, template.abs_distance)

    /*** Save best match.                                                   ***/
    if score < best_score then
        score ← best_score
        gesture_class ← template.gesture_class

return gesture_class
```

Algorithm B.6: JACKKNIFE-PREPARE (*trajectory*, *resample_rate*)

```
/*** Uniformly resample trajectory by setting σ² = 0.0, and store normalized vectors. ***/
prepared.pts ←STOCHASTIC-RESAMPLE(trajectory, resample_rate, σ² = 0.0)
prepared.vectors ← VECTORIZE(prepared.pts)

/*** Extract query extents.                                                ***/
minimum, maximum ← BOUNDING-BOX(trajectory)
extents ← maximum − minimum
prepared.extents ←NORMALIZED(extents)

/*** Make vector of distances traveled by component                        ***/
m ← |sample.pts[0]|
abs_distance ← ZERO-VECTOR(m)

for (i ← 1; i < resample_rate; i ← i + 1) do
    for (c ← 0; c < m; c ← c + 1) do
        delta ← sample.pts[i][c] − sample.pts[i − 1][c]
        abs_distance[c] ← abs_distance[c]+ABS(delta[c])

prepared.abs_distance ← abs_distance
return prepared
```

Algorithm B.7: DTW-IP (*query_vectors*, *template_vectors*, *ratio*)

```
/*** Get length of query, which must match length of template.                    ***/
n ← |query_vectors|

/*** Calculate size of Sakoe-Chiba band.                                          ***/
band ← ROUND(n × ratio)

/*** Allocate and initialize an n × n cost matrix.                                ***/
/*** Matrix will be infinity everywhere except upper left corner.                ***/
cost ← MATRIX(n+1, n+1, init_value = ∞)
cost[0, 0] ← 0

/*** Fill in cost matrix.                                                         ***/
for (q ← 1; q ≤ n; q ← q+1) do

    /*** Restrict path through cost matrix using a Sakoe-Chiba band.              ***/
    minimum ← MAX(1, q − band)
    maximum ← MIN(q + band, n)

    for (t ← minimum; t ≤ maximum; t ← t+1) do

        /*** Measure similarity of template and query vectors being matched.      ***/
        d ← 1.0− DOT-PRODUCT(query_vectors[q − 1], template_vectors[t − 1])
        d ← MIN(2.0, MAX(0.0, d))

        /*** Select best path through matrix and extend.                          ***/
        best_path ← MIN(cost[q − 1, t], cost[q − 1, t − 1], cost[q, t − 1])
        cost[q, t] ← best_path + d

return cost[n, n]
```

B.3 Gesture Path Stochastic Resampling

Algorithm B.8: STANDARD-GPSR (*strokes*)

```
/*** Creates a synthetic sample from the given sample.                            ***/
unistroke ←MAKE-STOCHASTIC-UNISTROKE(strokes)
N ←GPSR-OPTIMAL-N(unistroke)
x ← 2
σ² ← 0.25
unistroke ←GPSR(unistroke, N, x, σ²)
multistroke ← MAKE-MULTISTROKE(unistroke)
return multistroke
```

Algorithm B.9: MAKE-STOCHASTIC-UNISTROKE (*strokes*)

```
/*** Permute strokes with Fisher-Yates shuffle,                    ***/
/*** and handle stroke direction invariance.                       ***/
for i ← |strokes| − 1 downto 0 do
    if randomize strokes then
        j ← RAND-INT(min = 0, max = i)
        SWAP(strokes[i], strokes[j])
        if Rand-Float(min=0, max=1) < 0.5 then
            REVERSE(strokes[i].points)

/*** Combine strokes into a unistroke.                             ***/
ret = {}
for i ← 0 to |strokes| − 1 do
    for j ← 0 to |stroke[i].points| − 1 do
        APPEND(ret, strokes[i].points[j])

return ret
```

Algorithm B.10: GPSR-OPTIMAL-N (*points*)

```
/*** Computes the optimal stochastic resampling factor for 2D data (Chapter 8.  ***/
endPointsDist ← DISTANCE(points[0], points[|points| − 1])
pathLength ← PATH-LENGTH(points)
diagonal ← DIAGONAL(points)
density ← pathLength / diagonal
closedness ← 1 − endPointsDist / diagonal

/*** Compute the optimal N                                         ***/
N ← exp(1.67 + 0.29 ∗ density + 1.42 ∗ closedness)

if N < 16 then
    N ← 16

return N
```

Algorithm B.11: MAKE-MULTISTROKE (*points*)

```
/*** Converts a unistroke to a multistroke.                        ***/
multistroke ← {}, temp ← {}
id ← 0
foreach point in points do
    prev_id ← id
    id ← point.stroke_id

    /*** The point belongs to a stroke part                        ***/
    /*** if its id is an integer.                                  ***/
    if IsInteger(id) then
        if temp ≠ {} and prev_id ≠ id then
            APPEND(multistroke, temp)
            temp ← {}
        APPEND(temp, point);

/*** Append any remaining points.                                  ***/
if temp ≠ {} then
    APPEND(multistroke, temp);

return multistroke
```

Algorithm B.12: STOCHASTIC-RESAMPLE ($points$, n, σ^2)

```
/*** Generate n - 1 random intervals.                                    ***/
scale ← √(12 + σ²)
intervals ← []
total ← 0
for i ← 0 to n − 2 do
    interval ← 1 + RAND-FLOAT(min = 0, max = 1) ∗ scale
    APPEND(intervals, interval)
    total ← total + intervals_i

for i ← 0 to n − 2 do
    intervals[i] ← intervals[i] / total

/*** Perform resampling.                                                 ***/
pathDistance ←PATH-LENGTH(points)
cnt ← 0
I ← pathDistance ∗ intervals[cnt]
D ← 0
ret ← [points[0]]

foreach p[i] in points for i ≥ 1 do
    d ←DISTANCE(p[i], p[i − 1])
    if D + d ≥ I then
        t ←MIN(MAX( (I − D) / d, 0), 1)
        prev ← p[i − 1]]
        q ← (1 − t) ∗ prev + t ∗ p[i]
        APPEND(ret, q)
        INSERTAT(points, i, q)
        D ← 0
        cnt ← cnt + 1
        I ← pathDistance ∗ intervals[cnt]
    else
        D ← D + d

return ret
```

Algorithm B.13: GPSR ($points$, n, x, σ^2)

```
/*** Stochastically resample trajectory                                  ***/
points ← STOCHASTIC-RESAMPLE(points, n + x, σ²)

/*** Remove random points.                                               ***/
for i ← 0 to x − 1 do
    idx ← RAND-INT(min = 0, max = n − i)
    REMOVE(points, idx)

/*** Normalize vectors.                                                  ***/
m ← |points[0]|
syntheticSample ← [ZERO-VECTOR(m)]
for i ← 1 to |points| − 1 do
    vector ← points[i] − points[i − 1]
    NORMALIZE-VECTOR(vector)
    q ← syntheticSample[i − 1] + vector
    APPEND(syntheticSample, q);

return syntheticSample
```

B.4 Machete

Algorithm B.14: INITIALIZE-TEMPLATE($samplePts,\ \theta,\ \varepsilon$)

```
/*** Initialize house keeping data.  Note that row is a circular buffer over two CDP_ip rows.  ***/
template.buffer ← []
template.row ← [[], []]
template.T⃗ = []
template.startFrame ← 0
template.endFrame ← 0
template.currRowIdx ← 0
template.s1 ← 0
template.s2 ← 0
template.s3 ← 0

/*** Resample using Angular DP and build template and create initial CDP_ip matrix.  ***/
pts ← ANGULAR-DP (samplePts, ε)
N ← |pts|
for i ← 0 to N − 1 do
    elem.startFrame ← −1
    elem.endFrame ← −1
    elem.cumulativeCost ← 0
    elem.cumulativeLength ← 0
    elem.score ← ∞
    if i = 0 then elem.score ← (1 − cos(θ))^2

    PUSH-BACK(template.row[0], elem)
    PUSH-BACK(template.row[1], elem)

    if i > 0 then
        v⃗ ← pts[i] − pts[i − 1]
        PUSH-BACK(template.T⃗, v⃗/‖v⃗‖)

/*** Calculate correction factor values and weights.  ***/
f2l⃗ ← pts[N − 1] − pts[0]
diagLength ←DIAGONAL-LENGTH(pts)
length ←PATH-LENGTH(pts)
template.f2l⃗ ← f2l⃗/‖f2l⃗‖
template.openness ← ‖f2l⃗‖/length
template.w_closedness ← 1 − ‖f2l⃗‖/diagLength
template.w_f2l ← min(1, 2‖f2l⃗‖/diagLength)
return template
```

Algorithm B.15: ANGULAR-DP ($trajectory$, ε)

```
/*** Determine threshold that stops recursion.  ***/
diagLength ←DIAGONAL-LENGTH(trajectory)
ε ← diagLength × ε

/*** Recursively find most descriptive points.  ***/
newPts ← {}
N ← |trajectory|
PUSH-BACK(newPts, trajectory[0])
ANGULAR-DP-RECURSIVE(trajectory, 0, N − 1, newPts, ε)
PUSH-BACK(newPts, trajectory[N − 1])
return newPts
```

Algorithm B.16: ANGULAR-DP-RECURSIVE ($trajectory$, $start$, end, $newPts$, ε)

```
/*** Base case.                                                          ***/
if start + 1 ≥ end then return

/*** Calculate distance from every point in [start+1, end-1]            ***/
/*** to the line defined by trajectory[start] to trajectory[end].       ***/
AB ← trajectory[end] − trajectory[start]
denom ← ⟨AB, AB⟩
if denom = 0 then return

largest ← ε
selected ← −1

for idx ← start + 1 to end − 1 do
    /*** Project point onto line segment AB.                            ***/
    AC ← trajectory[idx] − trajectory[start]
    numer ← ⟨AB, AC⟩
    d2 ← ⟨AC, AC⟩ − (numer^2 / denom)

    /*** Get vectors made by end points and this point.                 ***/
    v1 ← trajectory[idx] − trajectory[start]
    v2 ← trajectory[end] − trajectory[idx]
    l1 ← ||v1||
    l2 ← ||v2||
    if l1 × l2 = 0 then continue

    /*** Calculate weighted distance and save if it's the best so far.  ***/
    d ← ⟨v1, v2⟩ / (l1 × l2)
    distance ← sqrt(d2 × ACOS(d)/π)
    if distance ≥ largest then
        largest ← distance
        selected ← idx

if selected = −1 then return

/*** If we split the subsequence, then recurse into each half.  Also save the split point.  ***/
ANGULAR-DP-RECURSIVE(trajectory, start, selected, newPts, ε)
PUSH-BACK(newPts, trajectory[selected])
ANGULAR-DP-RECURSIVE(trajectory, selected, end, newPts, ε)
```

Algorithm B.17: CALCULATE-CORRECTION-FACTORS($template$, $cdpElem$)

```
/*** Calculate the first-to-last vector, then the closeness and first-to-last correction factors.  ***/
f2l ← template.buffer[cdpElem.endFrame] − template.buffer[cdpElem.startFrame]
f2lLength ← ||f2l||
openness ← f2lLength / cdpElem.cumulativeLength

cf_openness ← 1 + template.w_closedness × ( max(openness, template.openness) / min(openness, template.openness) − 1 )
cf_openness = min(2, cf_closeness)

cf_f2l ← 1 + 1/2 × template.w_f2l × ( 1 − ⟨ f2l / f2lLength, template.f2l ⟩ )
cf_f2l = min(2, cf_f2l)
return (cf_openness × cf_f2l)
```

Algorithm B.18: CONSUME-INPUT (*template*, $\boldsymbol{x}$, *frameNumber*)

```
/*** Add input to buffer.                                                          ***/
PUSH-BACK(template.buffer, x)

/*** Filter out inconsequential movement.  We use δ = 1 for mouse data and zero otherwise.  ***/
length ← ||x − template.prev||
if length < δ then return

/*** Convert to direction vector.                                                  ***/
x⃗ ← (x − template.prev) / length
template.prev ← x

/*** We store two rows of matrix data, accessed as a circular buffer.              ***/
prevRow ← template.row[template.currRowIdx]
template.currRowIdx ← (template.currRowIdx + 1) mod 2
currRow ← template.row[template.currRowIdx]
currRow[0].startFrame ← frameNumber

/*** Update current row with new input.                                            ***/
T⃗ ← template.T⃗
T_N ← |T⃗|

for col ← 1 to T_N do

    /*** Determine which one of three paths to extend.                             ***/
    best ← currRow[col − 1]
    path2 ← prevRow[col − 1]
    path3 ← prevRow[col]

    if path2.score ≤ best.score then best ← path2
    if path3.score ≤ best.score then best ← path3

    /*** Extend selected path through current column.                              ***/
    localCost ← length × (1 − ⟨x⃗, t⃗[col − 1]⟩)^2
    currRow[col].startFrame ← best.startFrame
    currRow[col].endFrame ← frameNumber
    currRow[col].cumulativeCost ← best.cumulativeCost + localCost
    currRow[col].cumulativeLength ← best.cumulativeLength + length
    currRow[col].score ← currRow[col].cumulativeCost / currRow[col].cumulativeLength

cf ← CALCULATE-CORRECTION-FACTORS(template, currRow[T_N])
correctedScore ← cf × currRow[T_N].score

/*** Determine if we should call underlying recognizer.                            ***/
template.doCheck ← false
template.total ← template.total + currRow[T_N].score
template.n ← template.n + 1
template.s1 ← template.s2
template.s2 ← template.s3
template.s3 ← correctedScore

/*** If new low, save segmentation information.                                    ***/
if template.s3 < template.s2 then
    template.startFrame = currRow[T_N].startFrame
    template.endFrame = currRow[T_N].endFrame
    return

/*** If previous frame is a minimum below the threshold, trigger check             ***/
μ = template.total / (2 × template.n)
template.doCheck ← (template.s2 < μ  &  template.s2 < template.s1  &  template.s2 < template.s3)
```

B.5 Voight-Kampff-Machine

Algorithm B.19: VOIGHT-KAMPFF-MACHINE (*recognizer*, T, β)

```
/*** Given a recognizer trained with T training samples per gesture class, estimate a rejection threshold
     that maximizes the F_β-score.                                                                  ***/

/*** References to training data and samples prepared for mincing.                                  ***/
samples ← GET-TRAINING-SAMPLES(recognizer)
sampleCnt ← LENGTH(samples)
preparedSamples ← []

for (i ← 0; i < sampleCnt; i ← i+1) do
    prepared ← MINCER-PREPARE(samples[i])
    APPEND(preparedSamples, prepared)

/*** Place to store and track score distributions                                                   ***/
pos ← []
neg ← []
iterations ← 10000

/*** Go through each training sample and generate distributions.                                    ***/
for (tidx ← 0; tidx < sampleCnt; tidx ← tidx+1) do

    /*** Use mincer to generate and score negatives samples.                                        ***/
    for (j ← 0; j < iterations; j ← j+1) do
        mincedSample ← MINCE(preparedSamples, tidx)
        score ← recognizer.MEASURE(mincedSample)
        APPEND(neg, score)

    /*** Use GPSR to generate and scrore positive samples.                                          ***/
    gpsrN ← GPSR-OPTIMAL-N(samples[tidx])
    gpsrR ← 5
    for (j ← 0; j < iterations; j ← j+1) do
        positiveSample ← GPSR(samples[tidx], gpsrN, gpsrR)
        score ← recognizer.MEASAURE(positiveSample)
        APPEND(pos, score)

/*** Estimate and scale the rejection threshold.                                                    ***/
d ← 6
τ ← ESTIMATE-REJECTION-THRESHOLD(pos, neg, β)
λ ←ESTIMATE-ADJUSTMENT(d, τ, T, β)
threshold ← τ × λ

/*** Return global rejection threshold.                                                            ***/
return threshold
```

Algorithm B.20: ESTIMATE-REJECTION-THRESHOLD (pos, neg, β)

```
/*** Given unique positive and negative score distributions, estimate the threshold that maximizes F_β. ***/
SORT(pos)
SORT(neg)

/*** Prepare local variables.                                                                          ***/
pcnt ← LENGTH(pos)
ncnt ← LENGTH(neg)
tp ← 0
fn ← pcnt
fp ← 0
bestFscore ← −∞
bestThreshold ← −∞
pidx ← 0
nidx ← 0

/*** Walk both sorted lists, estimating F_β at each step, and saving the best result.                  ***/
while pidx < pcnt and nidx < ncnt do

    /*** Select next smallest threshold.                                                               ***/
    threshold ← MIN(pos[pidx], neg[nidx])

    /*** Update tracking variables to reflect change in threshold.                                     ***/
    while pidx < pcnt and pos[pidx] ≤ threshold do
        tp ← tp + 1
        fn ← fn − 1
        pidx ← pidx + 1

    while nidx < ncnt and neg[nidx] ≤ threshold do
        fp ← fp + 1
        nidx ← nidx + 1

    /*** Estiamte F_β-score and save if it's the best.                                                 ***/
    F_β = ((β^2+1.0) tp) / ((β^2+1.0) tp + β^2 fn + fp)

    if F_β > bestFscore then
        bestFscore ← F_β
        bestThreshold ← threshold

return bestThreshold
```

Algorithm B.21: ESTIMATE-ADJUSTMENT (d, τ, ι, T, β)

```
/*** Given a hypersphere of dimensionality d, rejection threshold τ, inflation factor ι, and training set
     size T, find a threshold scaling factor λ that maximizes F_β.                                    ***/
rt ← τ // Radius around training space.
ri ← τ × ι // Radius around application space.

/*** Iterate over a range of possible adjustment factors.                                           ***/
for (λ ← .5; λ <= 2; λ ← λ + 0.05) do

    tp, fp, fn ← 0, 0, 0
    rmax ← max(ri, τ(1 + λ))
    bestFscore ← 0
    adjustment ← 0

    /*** Uniformly sample points in test space and classify them.                                   ***/
    for (i ← 0; i < 1e6; i ← i + 1) do

        test ← SAMPLESPHERE(d, rmax)
        inside ← EUCLIDEANDISTANCE(test) < ri // Is inside the application space?
        recognized ← FALSE

        /*** Sample one point for each training sample.                                             ***/
        for (t ← 0; t < T; t ← t + 1) do

            trainPt ← SAMPLESPHERE(d, rt)
            distance ← EUCLIDEANDISTANCE(trainPt − test)
            if distnace < rt × λ then
                recognized ← TRUE
                break

    /*** Eastimate F_β-score.                                                                       ***/
    if inside and recognized then tp ← tp + 1
    if inside and not recognized then tn ← tn + 1
    if not inside and recognized then tp ← tp + 1
    score ← FSCORE(β, tp, fp, tn)

    /*** Save adjustment if we found a new best score.                                              ***/
    if score > bestFscore then
        bestFscore ← score
        adjustment ← λ

return adjustment
```

Algorithm B.22: MINCER-PREPARE(*sample*)

```
/*** Move the points to the centroid and vectorize                                  ***/
prepared ← UNIFORM-RESAMPLE(sample.trajectory, n = 128)
minimum ← MINIMUM-POINT(prepared)
maximum ← MAXIMUM-POINT(prepared)

/*** Get Centroid                                                                   ***/
centroid ← prepared[0]
for (i ← 1; i < len(prepared); i ← i + 1) do
    centroid ← centroid + prepared[i]
centroid ← centroid / len(prepared)

/*** Move centroid to origin and scale by the maximum component.                    ***/
width ← maximum − minimum
scale ← MAXIMUM-COMPONENT(width)

for (i ← 0; i < len(prepared); i ← i + 1) do
    prepared[i] ← prepared[i] − centroid
    prepared[i] ← prepared[i] / scale

/*** Vectorize the translated and scaled points                                     ***/
for (i ← 0; i < len(prepared) − 1; i ← i + 1) do
    prepared[i] ← prepared[i + 1] − prepared[i]
prepared.REMOVE-LAST()

return prepared
```

Algorithm B.23: MINCE(*targetIdx*)

```
/*** Given a set of training samples that have been centered, scaled, and spatially resampled, mince the
     target_idx sample.                                                                                  ***/


/*** Select a random sample that does not belong to the gesture class of the prepared samples of target
     index                                                                                               ***/
gestureClassName ← trainingSamples[targetIdx].gname;
cnt ← LENGTH(trainingSamples)
repeat
    otherIdx ← RANDOMINT(0, cnt − 1)
until gestureClassName ≠ trainingSamples[otherIdx].gname

/*** Select subset of gesture to replace.                                                              ***/
/*** Assume all samples have been resampled to same length.                                            ***/
N ← LENGTH(preparedSamples[0])
threshold ← N/3
while true do
    idx1 ←RANDOMINT(0, N − 1)
    idx2 ←RANDOMINT(0, N − 1)
    width ←ABS(idx2 − idx1) + 1

    if width < threshold then continue
    break

/*** Make zero point.  There are m components per point.                                               ***/
m ← LENGTH(preparedSamples[0][0])
pt ← ZERO-VECTOR(m)
mincedSample ← []
APPEND(mincedSample, pt)

/*** Move first part of target into minced sample.                                                     ***/
for (i ← 0; i < min(idx1, idx2); i ← i + 1) do
    pt ← pt + preparedSamples[targetIdx][i]
    APPEND(mincedSample, pt)

/*** Copy part of other trajectory into minced sample.                                                 ***/
ii ← idx1
if ii ≤ idx2 then
    while ii ≤ idx2 do
        pt ← pt + preparedSamples[otherIdx][ii]
        APPEND(mincedSample, pt)
        ii ← ii + 1
else
    while ii ≥ idx2 do
        pt ← pt + preparedSamples[otherIdx][ii]
        APPEND(mincedSample, pt)
        ii ← ii − 1

/*** Move last part of target into minced sample.                                                      ***/
for (i ← max(idx1, idx2) + 1; i < N; i ← i + 1) do
    pt ← pt + preparedSamples[targetIdx][i]
    APPEND(mincedSample, pt)

return mincedSample
```

Algorithm B.24: MACHETE-GPSR-OPTIMAL-N(*trajectory*)

```
/*** Get the optimal resample count for GPSR                                    ***/

resampled ← UNIFORM-RESAMPLE(trajectory, 64)
prev ← resampled[0]
curr ← resampled[0]

/*** Calculate the absolute curvature of a gesture (angle).                     ***/
angle ← 0
for (i ← 1; i < len(resampled); i ← i + 1) do
    prev ← curr
    curr ← resampled[i] − resampled[i − 1]
    curr ← NORMALIZE-VECTOR(curr)
    if i < 2 then
        continue

    /*** Update the total angle of the turn                                     ***/
    dot ← max(−1.0, min(1.0, DOT-PRODUCT(prev, curr)))
    theta ← ARCCOS(dot)
    angle ← angle + theta

/*** Calculate the ratio of the path length to bounding box diagonal (density). ***/
minimum ← MINIMUM-POINT(resampled)
maximum ← MAXIMUM-POINT(resampled)
diagonal ← DISTANCE(maximum − minimum)
pathLength ← PATH-LENGTH(trajectory)
density ← pathLength/diagonal

/*** Get the precomputed coefficients                                           ***/
gpsrCoeffs ← GET-GPSR-COEFFICIENTS()

/*** Use the coefficients, angle, and density to find the optimal resample count N ***/
ret ← gpsrCoeffs.intercept
ret ← ret + gpsrCoeffs.densityCo ∗ density
ret ← ret + gpsrCoeffs.angleCo ∗ angle
ret ← ret + gpsrCoeffs.densityAngleCo ∗ density ∗ angle

return ROUND(ret)
```

B.6 Prototype Selection

Algorithm B.25: RANDOM-MUTATION-HILL-CLIMB (*Samples*, k)

```
for i ← 0 to k do
    Fittest[i] ← RANDOM-SAMPLE(Samples)
FittestRecognizer ← GENERATE-RECOGNIZER(Fittest)
count ← 0
max ← 32 * k
TestMutations ← 256
while count < max do
    Alternate ← Fittest
    MUTATE-ONE(Samples,Alternate)
    AlternateRecognizer ← GENERATE-RECOGNIZER(Alternate)
    c1 ← 0
    c2 ← 0
    n ← 0
    for i ← 0 to TestMutations do
        for j ← 0 to Gesture-Type-Count(Samples) do
            S ← RANDOM-SAMPLE-OF-TYPE(Samples, j)
            c1 ← c1+IS-RECOGNIZED(FittestRecognizer, S)
            c2 ← c2+IS-RECOGNIZED(AlternateRecognizer, S)
        k ← FIND-CRITICAL-VALUE(c1, c2, n)
        if c2 < k then
            break
    if c2 > c1 then
        Fittest ← Alternate
        FittestRecognizer ← AlternateRecognizer
return Fittest
```

Algorithm B.26: FIND-CRITICAL-VALUE (*correct*1, *correct*2, n)

```
α ← 0.8
psuccess ← correct1/n
pfail ← 1.0 − psuccess
ps ← psuccess^n
pf ← 1.0
comb ← 1.0
for k ← n to 0 do
    if α ≤ 0 then
        break
    α ← α − comb * ps * pf
    comb ← (comb * k)/(n − k + 1)
    ps ← ps/psuccess
    pf ← pf * pfail
return k
```

APPENDIX C: COPYRIGHT RELEASE AND PERMISSION MATERIAL

IDENTIFICATION

Paper Number (also write on every page of this release): 144

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

Paper Authors: Eugene M. Taranta II Joseph J. LaViola Jr.

CONTACT

Please provide full contact information for the author or other person who will be responsible for the paper during the publication process. This person should be available and able to address any problems that might arise during the printing of the paper.

Name: Eugene M. Taranta II

Address 1: 301 Capron Ash Loop

Address 2:

Address 3:

City: Casselberry

State/Province: FL

Country and Postal/Zip Code: USA 32707

Phone number: (407) 461-3571

Fax number:

Email: etaranta@gmail.com

IMAGES AND SUPPLEMENTARY FILES Paper 144

(CIRCLE **have** or **have not** in the following, add descriptions, sign below)
I **have** / (**have not**) uploaded the following files for electronic dissemination by the Canadian Human-Computer Communications Society with the same non-exclusive publication rights as the electronic version of the paper: (describe files, if any)

Signature: Eugene M. Taranta II

(CIRCLE **may** or **may not** in the following, add descriptions, sign below)
The images in the paper (**may**) / **may not** be considered for publication on the cover of the Proceedings. Note: please give permission if you can. If only *some* images in your paper cannot be reproduced outside the paper (for instance, pictures of people in experimental situtations), give permission as above and describe the exceptions (the images which CANNOT be on the cover) below:

Signature: Eugene M. Taranta II

Paper 144

(CIRCLE **have** or **have not** in the following, add descriptions, sign below)
I **have** / (**have not**) uploaded to the final submission site (PCS) additional images (images different from those in the actual paper, or better-quality versions of images in the paper) for consideration for the cover of Graphics Interface 2015. If I have uploaded such images, the proceedings editor has my permission to publish them. The images are described as follows:

Signature: Eugene M. Taranto II

Paper 144

PAGE CHARGES

My paper contains 8 pages, of which 2 should be printed in colour.

The following pages should be printed in colour: 5, 6

The following charges are in Canadian dollars.

Pages in excess of eight:	⌈ 0 /2⌉ × $200	=	$ 0
First **two** colour pages:	2 × $0	=	$ 0
Subsequent colour pages:	0 × $50	=	$ 0
Total page charges:			$ 0

METHOD OF PAYMENT

Page charges will be included in the registration fee.

Ontario HST (harmonized sales tax) will be included in the page charges. The CHCCS/ SCDHM HST/GST Registration number is 12617 0760 RT0001. Receipts will be provided to the attending author at the conference.

Paper 144

COPYRIGHT AUTHORIZATION

- I give permission for the Canadian Human-Computer Communications Society/ Socit canadienne du dialogue humain-machine (CHCCS/ SCDHM) to publish my paper in print in the Proceedings of Graphics Interface 2015.

- I give permission for the Canadian Human-Computer Communications Society/ Socit canadienne du dialogue humain-machine (CHCCS/ SCDHM) to publish my paper electronically on the Graphics Interface website.

- I give permission for the Canadian Human-Computer Communications Society/ Socit canadienne du dialogue humain-machine (CHCCS/ SCDHM) to publish my paper electronically in the ACM Digital Library, and on any future electronic distribution media (including on USBs or DVDs) along with any additional electronic material submitted by me, as described above.

- I promise that I or one of my coauthors will come to the Graphics Interface 2015 conference to present the paper.

- These rights of publication and which I grant to the Canadian Human-Computer Communications Society/ Socit canadienne du dialogue humain-machine (CHCCS/ SCDHM) are non-exclusive.

Printed Name: Eugene M. Taranta II

Signature: Eugene M. Taranta II

Date: March 15 2015

ACM Copyright and Audio/Video Release

Title of the Work: Jackknife: A Reliable Recognizer with Few Samples and Many Modalities
Submission ID:pn4460
Author/Presenter(s): Eugene M. Taranta II (Univ. of Central Florida); Amirreza Samiei (Univ. of Central Florda); Mehran Maghoumi (Univ. of Central Florida); Pooya Khaloo (Univ. of Central Florida); Corey R Pittman (Univ. of Central Florida); Joseph J LaViola Jr. (Univ. of Central Florida)
Type of material: Paper

Publication and/or Conference Name: CHI'17: CHI Conference on Human Factors in Computing Systems Proceedings

I. Copyright Transfer, Reserved Rights and Permitted Uses

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new Authorized ACM TeX template .cls version 2.8, automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

Please copy and paste the following code snippet into your TeX file between \begin{document} and \maketitle, either after or before CCS codes.

```
\CopyrightYear{2017}
\setcopyright{acmcopyright}
\conferenceinfo{CHI 2017,}{May 06-11, 2017, Denver, CO, USA}
\isbn{978-1-4503-4655-9/17/05}\acmPrice{\$15.00}
\doi{http://dx.doi.org/10.1145/3025453.3026002}
```

If you are using the ACM Microsoft Word template, or still using an older version of the ACM TeX template, or the current versions of the ACM SIGCHI, SIGGRAPH, or SIGPLAN TeX templates, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

CHI 2017, May 06-11, 2017, Denver, CO, USA
© 2017 ACM. ISBN 978-1-4503-4655-9/17/05...$15.00
DOI: http://dx.doi.org/10.1145/3025453.3026002

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library

☑ A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable

(Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government? ○ Yes ◉ No

---

II. PERMISSION FOR CONFERENCE TAPING AND DISTRIBUTION

Audio/Video Release
* Your Audio/Video Release is conditional upon you agreeing to the terms set out below.

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ◉ Yes ○ No

III. Auxiliary Material

Do you have any Auxiliary Materials? ◉ Yes ○ No

* Your Auxiliary Materials Release is conditional upon you agreeing to the terms set out below.

[Defined as additional files, video or software and executables that are not submitted for review and publication as an integral part of the Work but are supplied by the author as useful resources.] I hereby grant ACM permission to serve files containing my Auxiliary Material from the ACM Digital Library. I hereby represent and warrant that my Auxiliary (software) does not knowingly and surreptitiously incorporate malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software.

☑ I agree to the above Auxiliary Materials permission statement.

☐ This software is knowingly designed to illustrate technique(s) intended to defeat a system's security. The code has been explicitly documented to state this fact.

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

◉ We/I have not used third-party material.

○ We/I have used third-party materials and have necessary permissions.

V. Artistic Images
If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.
◉ We/I do not have any artistic images.
○ We/I have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants

DATE: **01/02/2017** sent to etaranta@gmail.com at **10:01:15**

ACM Copyright Form and Audio/Video Release

Title of the Work: Machete: Easy, Efficient, and Precise Continuous Custom Gesture Segmentation

Author/Presenter(s): Mr. Eugene Taranta (University of Central Florida);Mr. Corey Pittman (University of Central Florida);Mr. Mehran Maghoumi (University of Central Florida);Mr. Mykola Maslych (University of Central Florida);Miss Yasmine Moolenaar (University of Central Florida);Prof. Joseph LaViola Jr (U Central Florida)

Type of material: Paper

Publication: ACM Transactions on Computer-Human Interaction

I. Copyright Transfer, Reserved Rights and Permitted Uses

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, or (3) any repository legally mandated by an agency funding the research on which the Work is based.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

(x)If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

When preparing your paper for submission using the ACM templates, you will need to include the rights management and bibstrip text blocks below to the lower left hand portion of the first page. As this text will provide rights information for your paper, please make sure that this text is displayed and positioned correctly when you submit your manuscript for publication.

Authors should understand that consistent with ACM's policy of encouraging dissemination of information, each work published by ACM appears with a copyright and the following notice:

> *If you are using Authorized ACM TeX templates, the following code will generate the proper statements based on your rights choices. Please copy and paste it into your TeX file between \begin{document} and \maketitle, either after or before CCS codes.*
>
> \setcopyright{acmcopyright}
> \acmJournal{TOCHI}
> \acmYear{2020} \acmVolume{1} \acmNumber{1} \acmArticle{1}
> \acmMonth{1} \acmPrice{15.00}\acmDOI{10.1145/3428068}
>
> *If you are using Word, copy and paste these words in the space provided at the bottom of your first page:*
>
> 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.
>
> Copyright © ACM 2020 1073-0516/2020/MonthOfPublication - ArticleNumber $15.00
> https://doi.org/10.1145/3428068

NOTE: DOIs will be registered and become active shortly after publication in the ACM

Digital Library

☑ A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government?
○ Yes ◉ No
Country:

II. PERMISSION FOR CONFERENCE TAPING AND DISTRIBUTION (Check A and, if applicable, B)
A. Audio /Video Release

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately by itself as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ◉ Yes ○ No

III. Auxiliary Materials, not integral to the Work

* Your Auxiliary Materials Release is conditional upon you agreeing to the terms set out below.

[Defined as additional files, including software and executables that are not submitted for review and publication as an integral part of the Work but are supplied by the author as useful resources for the reader.]

I hereby grant ACM permission to serve files containing my Auxiliary Material from the ACM Digital Library. I hereby represent and warrant that any of my Auxiliary Materials do not knowingly and surreptitiously incorporate malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software.

☐ I agree to the above Auxiliary Materials permission statement.

☐ This software is knowingly designed to illustrate technique(s) intended to defeat a

system's security. The code has been explicitly documented to state this fact.

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

◉ We/I have not used third-party material.
○ We/I have used third-party materials and have necessary permissions.

V. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part IV and be sure to include a notice of copyright with each such image in the paper.

◉ We/I do not have any artistic images.
○ We/I have have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants

DATE: **10/03/2020** sent to etaranta@gmail.com at **16:10:22**

ACM Copyright and Audio/Video Release

Title of the Work: A Rapid Prototyping Approach to Synthetic Data Generation For Improved 2D Gesture Recognition
Submission ID:uist2105
Author/Presenter(s): Eugene M. Taranta II (Univ. of Central Florida); Mehran Maghoumi (Univ. of Central Florida); Corey R Pittman (Univ. of Central Florida); Joseph J LaViola Jr. (Univ. of Central Florida)
Type of material: Paper

Publication and/or Conference Name: UIST '16: The 29th Annual ACM Symposium on User Interface Software and Technology Proceedings

I. Copyright Transfer, Reserved Rights and Permitted Uses

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new Authorized ACM TeX template .cls version 2.8, automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

Please copy and paste the following code snippet into your TeX file between \begin{document} and \maketitle, either after or before CCS codes.

```
\CopyrightYear{2016}
\setcopyright{acmcopyright}
\conferenceinfo{UIST 2016,}{October 16-19, 2016, Tokyo, Japan}
\isbn{978-1-4503-4189-9/16/10}\acmPrice{\$15.00}
\doi{http://dx.doi.org/10.1145/2984511.2984525}
```

If you are using the ACM Microsoft Word template, or still using an older version of the ACM TeX template, or the current versions of the ACM SIGCHI, SIGGRAPH, or SIGPLAN TeX templates, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

UIST 2016, October 16-19, 2016, Tokyo, Japan

DOI: http://dx.doi.org/10.1145/2984511.2984525

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library

☑ A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable

(Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government? ○ Yes ◉ No
Country:

II. PERMISSION FOR CONFERENCE TAPING AND DISTRIBUTION

Audio/Video Release
* Your Audio/Video Release is conditional upon you agreeing to the terms set out below.

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ◉ Yes ○ No

III. Auxiliary Material

Do you have any Auxiliary Materials? ◉ Yes ○ No

* Your Auxiliary Materials Release is conditional upon you agreeing to the terms set out below.

[Defined as additional files, video or software and executables that are not submitted for review and publication as an integral part of the Work but are supplied by the author as useful resources.] I hereby grant ACM permission to serve files containing my Auxiliary Material from the ACM Digital Library. I hereby represent and warrant that my Auxiliary (software) does not knowingly and surreptitiously incorporate malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software.

☑ I agree to the above Auxiliary Materials permission statement.

☐ This software is knowingly designed to illustrate technique(s) intended to defeat a system's security. The code has been explicitly documented to state this fact.

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

◉ We/I have not used third-party material.
○ We/I have used third-party materials and have necessary permissions.

V. Artistic Images
If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.
◉ We/I do not have any artistic images.
○ We/I have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants

Funding Agents

1. National Science Foundation award number(s): IIS-0845921

DATE: **08/09/2016** sent to etaranta@gmail.com at **21:08:18**

ACM Copyright and Audio/Video Release

Title of the Work: A $-Family Friendly Approach to Prototype Selection
Submission ID:iuifp320
Author/Presenter(s): Corey Pittman (Univ. of Central Florida); Eugene Taranta II (Univ. of Central Florida); Joseph LaViola Jr. (Univ. of Central Florida)
Type of material: Short Paper

Publication and/or Conference Name: IUI'16: 21st International Conference on Intelligent User Interfaces Proceedings

I. Copyright Transfer, Reserved Rights and Permitted Uses

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, or (3) any repository legally mandated by an agency funding the research on which the Work is based.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new Authorized ACM TeX template .cls version 2.8, automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

Please copy and paste the following code snippet into your TeX file between \begin{document} and \maketitle, either after or before CCS codes.

```
\CopyrightYear{2016}
\setcopyright{acmcopyright}
\conferenceinfo{IUI'16,}{March 07-10, 2016, Sonoma, CA, USA}
\isbn{978-1-4503-4137-0/16/03}\acmPrice{\$15.00}
\doi{http://dx.doi.org/10.1145/2856767.2856808}
```

If you are using the ACM Microsoft Word template, or still using an older version of the ACM TeX template, or the current versions of the ACM SIGCHI, SIGGGRAPH, or SIGPLAN TeX templates, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

IUI'16, March 07-10, 2016, Sonoma, CA, USA

DOI: http://dx.doi.org/10.1145/2856767.2856808

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library

☑ A. Assent to Assignment. I hereby represent and warrant that I am the sole owner

(or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government?
○ Yes ◉ No
Country:

II. PERMISSION FOR CONFERENCE TAPING AND DISTRIBUTION
Audio /Video Release

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ◉ Yes ○ No

III. Auxiliary Material
Any additional materials, including software or other executables that are not submitted for review and therefore not an integral part of the work, but are included for publication.

Do you have any Auxiliary Materials? ○ Yes ◉ No

I hereby grant ACM permission to serve files containing my Auxiliary Material from the ACM Digital Library. I hereby represent and warrant that my Auxiliary Materials do not knowingly and surreptitiously incorporate malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software.

☐ I agree to the above Auxiliary Materials permission statement.

☐ This software is knowingly designed to illustrate technique(s) intended to defeat a system's security. The code has been explicitly documented to state this fact.

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted

music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

(•) We/I have not used third-party material.
( ) We/I have used third-party materials and have necessary permissions.

V. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.

(•) We/I do not have any artistic images.
( ) We/I have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants

Funding Agents

1. NSF CAREER award award number(s): IIS-0845921

DATE: **01/03/2016** sent to cpittman@knights.ucf.edu; etaranta@knights.ucf.edu at **12:01:46**

ACM Copyright and Audio/Video Release

Title of the Work: Moving Toward an Ecologically Valid Data Collection Protocol for 2D Gestures in Video Games
Submission ID: pn4579
Author/Presenter(s): Eugene Matthew Taranta II; Corey Richard Pittman; Jack P. Oakley; Mykola Maslych; Mehran Maghoumi; Joseph LaViola
Type of material: Full Paper

Publication and/or Conference Name: CHI '20: CHI Conference on Human Factors in Computing Systems Proceedings

I. Copyright Transfer, Reserved Rights and Permitted Uses

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work. (x) If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new ACM Consolidated TeX template Version 1.3 and above automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

Please put the following LaTeX commands in the preamble of your document - i.e., before \begin{document}:

```
\copyrightyear{2020}
\acmYear{2020}
\setcopyright{acmcopyright}
\acmConference[CHI '20]{CHI Conference on Human Factors in Computing Systems}{April 25--30, 2020}{Honolulu, HI, USA}
\acmBooktitle{CHI Conference on Human Factors in Computing Systems (CHI '20), April 25--30, 2020, Honolulu, HI, USA}
\acmPrice{15.00}
\acmDOI{10.1145/3313831.3376417}
\acmISBN{978-1-4503-6708-0/20/04}
```

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

If you are using the ACM Interim Microsoft Word template, or still using or older versions of the ACM SIGCHI template, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

CHI '20, April 25–30, 2020, Honolulu, HI, USA

ACM ISBN 978-1-4503-6708-0/20/04...$15.00
https://doi.org/10.1145/3313831.3376417

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library. Once you have your camera ready copy ready, please send your source files and PDF to your event contact for processing.

☑ A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government? ○ Yes ◉ No

---

II. Permission For Conference Recording and Distribution

* Your Audio/Video Release is conditional upon you agreeing to the terms set out below.

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ◉ Yes ○ No

III. Auxiliary Material

Do you have any Auxiliary Materials? ◉ Yes ○ No

* Your Auxiliary Materials Release is conditional upon you agreeing to the terms set out below.

[Defined as additional files, video or software and executables that are not submitted for review and publication as an integral part of the Work but are supplied by the author as useful resources.] I hereby grant ACM permission to serve files containing my Auxiliary Material from the ACM Digital Library. I hereby represent and warrant that my Auxiliary (software) does not knowingly and surreptitiously incorporate malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized

access or to disable, erase or otherwise harm any computer systems or software.

☑ I agree to the above Auxiliary Materials permission statement.

☐ This software is knowingly designed to illustrate technique(s) intended to defeat a system's security. The code has been explicitly documented to state this fact.

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

◉ We/I have not used third-party material.
○ We/I have used third-party materials and have necessary permissions.

V. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.

◉ We/I do not have any artistic images.
○ We/I have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

☑ I agree to the Representations, Warranties and Covenants

DATE: **01/02/2020** sent to etaranta@gmail.com at **13:01:44**

APPENDIX D: IRB DOCUMENTATION

University of Central Florida Institutional Review Board
Office of Research & Commercialization
12201 Research Parkway, Suite 501
Orlando, Florida 32826-3246
Telephone: 407-823-2901 or 407-882-2276
www.research.ucf.edu/compliance/irb.html

Approval of Human Research

From: **UCF Institutional Review Board #1**
FWA00000351, IRB00001138

To: **Eugene Taranta** and Co-PIs: **Amirreza Samiei, Corey Pittman**

Date: **October 28, 2016**

Dear Researcher:

On 10/28/2016 the IRB approved the following human participant research until 10/27/2017 inclusive:

Type of Review:	Submission Response for UCF Initial Review Submission Form Expedited Review
Project Title:	Evaluating Jackknife Gesture Recognizer using Kinect and Leap Motion
Investigator:	Eugene Taranta
IRB Number:	SBE-16-12532
Funding Agency:	
Grant Title:	
Research ID:	N/A

The scientific merit of the research was considered during the IRB review. The Continuing Review Application must be submitted 30days prior to the expiration date for studies that were previously expedited, and 60 days prior to the expiration date for research that was previously reviewed at a convened meeting. Do not make changes to the study (i.e., protocol, methodology, consent form, personnel, site, etc.) before obtaining IRB approval. A Modification Form **cannot** be used to extend the approval period of a study. All forms may be completed and submitted online at https://iris.research.ucf.edu .

If continuing review approval is not granted before the expiration date of 10/27/2017, approval of this research expires on that date. When you have completed your research, please submit a Study Closure request in iRIS so that IRB records will be accurate.

Use of the approved, stamped consent document(s) is required. The new form supersedes all previous versions, which are now invalid for further use. Only approved investigators (or other approved key study personnel) may solicit consent for research participation. Participants or their representatives must receive a copy of the consent form(s).

All data, including signed consent forms if applicable, must be retained and secured per protocol for a minimum of five years (six if HIPAA applies) past the completion of this research. Any links to the identification of participants should be maintained and secured per protocol. Additional requirements may be imposed by your funding agency, your department, or other entities. Access to data is limited to authorized individuals listed as key study personnel.

In the conduct of this research, you are responsible to follow the requirements of the Investigator Manual.

On behalf of Sophia Dziegielewski, Ph.D., L.C.S.W., UCF IRB Chair, this letter is signed by:

Signature applied by Patria Davis on 10/28/2016 04:05:30 PM EDT

IRB Coordinator

University of Central Florida Institutional Review Board
Office of Research & Commercialization
12201 Research Parkway, Suite 501
Orlando, Florida 32826-3246
Telephone: 407-823-2901 or 407-882-2276
www.research.ucf.edu/compliance/irb.html

Approval of Human Research

From: **UCF Institutional Review Board #1**
FWA00000351, IRB00001138

To: **Eugene Taranta** and Co-PI: **Corey Pittman**

Date: **June 23, 2017**

Dear Researcher:

On 06/23/2017 the IRB approved the following human participant research until 06/22/2018 inclusive:

Type of Review:	UCF Initial Review Submission Form Expedited Review
Project Title:	Follow The Leader: An evaluation of continuous gesture recognition methods with different devices
Investigator:	Eugene Taranta
IRB Number:	SBE-17-13236
Funding Agency:	
Grant Title:	
Research ID:	N/A

The scientific merit of the research was considered during the IRB review. The Continuing Review Application must be submitted 30days prior to the expiration date for studies that were previously expedited, and 60 days prior to the expiration date for research that was previously reviewed at a convened meeting. Do not make changes to the study (i.e., protocol, methodology, consent form, personnel, site, etc.) before obtaining IRB approval. A Modification Form **cannot** be used to extend the approval period of a study. All forms may be completed and submitted online at https://iris.research.ucf.edu .

If continuing review approval is not granted before the expiration date of 06/22/2018, approval of this research expires on that date. When you have completed your research, please submit a Study Closure request in iRIS so that IRB records will be accurate.

Use of the approved, stamped consent document(s) is required. The new form supersedes all previous versions, which are now invalid for further use. Only approved investigators (or other approved key study personnel) may solicit consent for research participation. Participants or their representatives must receive a copy of the consent form(s).

All data, including signed consent forms if applicable, must be retained and secured per protocol for a minimum of five years (six if HIPAA applies) past the completion of this research. Any links to the identification of participants should be maintained and secured per protocol. Additional requirements may be imposed by your funding agency, your department, or other entities. Access to data is limited to authorized individuals listed as key study personnel.

In the conduct of this research, you are responsible to follow the requirements of the Investigator Manual.

On behalf of Sophia Dziegielewski, Ph.D., L.C.S.W., UCF IRB Chair, this letter is signed by:

Signature applied by Gillian Amy Mary Morien on 06/23/2017 10:21:40 AM EDT

IRB Coordinator

UNIVERSITY OF CENTRAL FLORIDA

Institutional Review Board
FWA00000351
IRB00001138
Office of Research
12201 Research Parkway
Orlando, FL 32826-3246

EXEMPTION DETERMINATION

September 23, 2019

Dear Eugene Taranta:

On 9/23/2019, the IRB determined the following submission to be human subjects research that is exempt from regulation:

Type of Review:	Initial Study, Exempt Category
Title:	Follow-the-Leader Protocol
Investigator:	Eugene Taranta
IRB ID:	STUDY00000893
Funding:	None
Grant ID:	None

This determination applies only to the activities described in the IRB submission and does not apply should any changes be made. If changes are made, and there are questions about whether these changes affect the exempt status of the human research, please contact the IRB. When you have completed your research, please submit a Study Closure request so that IRB records will be accurate.

If you have any questions, please contact the UCF IRB at 407-823-2901 or irb@ucf.edu. Please include your project title and IRB number in all correspondence with this office.

Sincerely,

Adrienne Showman
Designated Reviewer

LIST OF REFERENCES

[1] *PalmPilot Handbook*. 1997.

[2] Luke Ahearn. *Designing 3D games that sell!* Charles River Media, Inc., 2000.

[3] Mohammad Shukri Ahmad, Osman Kukrer, and Aykut Hocanin. Recursive inverse adaptive filtering algorithm. *Digital Signal Processing*, 21(4):491–496, 2011.

[4] Abdullah Almaksour, Eric Anquetil, Solen Quiniou, and Mohamed Cheriet. Personalizable pen-based interface using lifelong learning. In *2010 International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 188–193, nov 2010.

[5] Jonathan Alon, Vassilis Athitsos, Quan Yuan, and Stan Sclaroff. A unified framework for gesture recognition and spatiotemporal gesture segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 31(9):1685–1699, 2009.

[6] Leonardo Angelini, Francesco Carrino, Stefano Carrino, Maurizio Caon, Omar Abou Khaled, Jürgen Baumgartner, Andreas Sonderegger, Denis Lalanne, and Elena Mugellini. Gesturing on the steering wheel: a user-elicited taxonomy. In *Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 1–8. ACM, 2014.

[7] Lisa Anthony, YooJin Kim, and Leah Findlater. Analyzing user-generated youtube videos to understand touchscreen use by people with motor impairments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1223–1232. ACM, 2013.

[8] Lisa Anthony and Jacob 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.

[9] Lisa Anthony and Jacob 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.

[10] Caroline Appert and Shumin Zhai. Using strokes as command shortcuts: Cognitive benefits and toolkit support. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2289–2298, New York, NY, USA, 2009. ACM.

[11] Robert Arn, Pradyumna Narayana, Bruce Draper, Tegan Emerson, Michael Kirby, and Chris Peterson. Motion segmentation via generalized curvatures. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[12] Ahmad-Montaser Awal, Harold Mouchere, and Christian Viard-Gaudin. Towards handwritten mathematical expression recognition. In *Document Analysis and Recognition, 2009. ICDAR '09. 10th International Conference on*, pages 1046–1050, jul 2009.

[13] Clive Barker. *The hellbound heart*. HarperTorch, dec 1991.

[14] Ali Bashashati, Mehrdad Fatourechi, Rabab K Ward, and Gary E Birch. A survey of signal processing algorithms in brain–computer interfaces based on electrical brain signals. *Journal of Neural engineering*, 4(2):R32, 2007.

[15] Gustavo E.A.P.A. Batista, Xiaoyue Wang, and Eamonn J Keogh. A complexity-invariant distance measure for time series. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, chapter 60, pages 699–710. 2011.

[16] Leonard E Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3(1):1–8, 1972.

[17] Benjamin Bergen. Embodiment, simulation and meaning. *The Routledge handbook of semantics*, pages 142–157, 2015.

[18] Benjamin K Bergen. *Louder than words: The new science of how the mind makes meaning*. Basic Books (AZ), 2012.

[19] Zulfiquar Ali Bhotto and Andreas Antoniou. A family of shrinkage adaptive-filtering algorithms. *IEEE Transactions on Signal Processing*, 61(7):1689–1697, 2013.

[20] Ali Bigdelou, Loren Schwarz, Tobias Benz, and Nassir Navab. A flexible platform for developing context-aware 3d gesture-based interfaces. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, IUI '12, pages 335–336, New York, NY, USA, 2012. ACM.

[21] Kanad K Biswas and Saurav Kumar Basu. Gesture recognition using microsoft kinect®. In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pages 100–103. IEEE, 2011.

[22] Rachel Blagojevic, Samuel Hsiao-Heng Chang, and Beryl Plimmer. The power of automatic feature selection: Rubine on steroids. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 79–86, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[23] Aaron Bobick and James Davis. Real-time recognition of activity using temporal templates. In *Applications of Computer Vision, 1996. WACV'96., Proceedings 3rd IEEE Workshop on*, pages 39–42. IEEE, 1996.

[24] Aaron F Bobick and Andrew D Wilson. A state-based technique for the summarization and recognition of gesture. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 382–388. IEEE, 1995.

[25] Amlan Deep Borah, Abir J Mondal, Deboraj Muchahary, and Alak Majumder. FIR low pass filter design using craziness base particle swarm optimization technique. In *Communications and Signal Processing (ICCSP), 2015 International Conference on*, pages 870–874. IEEE, 2015.

[26] Jared N Bott, James G Crowley, and Joseph J LaViola Jr. Exploring 3d gestural interfaces for music creation in video games. In *Proceedings of the 4th international Conference on Foundations of Digital Games*, pages 18–25. ACM, 2009.

[27] Andreas Bulling, Daniel Roggen, and Gerhard Tröster. It's in your eyes: towards context-awareness and mobile hci using wearable eog goggles. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 84–93. ACM, 2008.

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

[29] Javier Cano, Juan-Carlos Perez-Cortes, Joaquim Arlandis, and Rafael Llobet. *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002 Windsor, Ontario, Canada, August 6–9, 2002 Proceedings*, chapter Training S, pages 548–556. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[30] Xiang Cao and Shumin Zhai. Modeling human performance of pen stroke gestures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1495–1504, New York, NY, USA, 2007. ACM.

[31] Fabio Marco Caputo, Pietro Prebianca, Alessandro Carcangiu, Lucio D Spano, and Andrea Giachetti. 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.

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

[33] Baptiste Caramiaux, Nicola Montecchio, Atau Tanaka, and Frédéric Bevilacqua. Adaptive gesture recognition with variation estimation for interactive systems. *ACM Trans. Interact. Intell. Syst.*, 4(4):18:1—-18:34, dec 2014.

[34] Alessandro Carcangiu. *Combining declarative models and computer vision recognition algorithms for stroke gestures*. PhD thesis.

[35] Géry Casiez, Nicolas Roussel, and Daniel Vogel. 1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2527–2530. ACM, 2012.

[36] Luigi Cattaneo and Giacomo Rizzolatti. The mirror neuron system. *Archives of neurology*, 66(5):557–560, 2009.

[37] Edwin Chan, Teddy Seyed, Wolfgang Stuerzlinger, Xing-Dong Yang, and Frank Maurer. User elicitation on single-hand microgestures. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3403–3414. ACM, 2016.

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

[39] Salman Cheema and Joseph J LaViola Jr. Physicsbook: a sketch-based interface for animating physics diagrams. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pages 51–60. ACM, 2012.

[40] Salman Cheema and Joseph J LaViola. Wizard of wii: toward understanding player experience in first person games with 3d gestures. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 265–267. ACM, 2011.

[41] Dali Chen, Dingyu Xue, and Feng Pan. An improved median filter based on automatic parameter tuning approach. In *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pages 1305–1309. IEEE, 2007.

[42] Jessie YC Chen and Jennifer E Thropp. Review of low frame rate effects on human performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(6):1063–1076, 2007.

[43] Yineng Chen, Xiaojun Su, Feng Tian, Jin Huang, Xiaolong Luke Zhang, Guozhong Dai, and Hongan Wang. Pactolus: A method for mid-air gesture segmentation within emg. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1760–1765. ACM, 2016.

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

[45] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, oct 2014. Association for Computational Linguistics.

[46] Morten H Christiansen and Simon Kirby. Language evolution: Consensus and controversies. *Trends in cognitive sciences*, 7(7):300–307, 2003.

[47] Rishabh Dabral, Anurag Mundhada, Uday Kusupati, Safeer Afaque, and Arjun Jain. Structure-aware and temporally coherent 3d human pose estimation. *arXiv preprint arXiv:1711.09250*, 2017.

[48] Nasser H Dardas and Nicolas D Georganas. Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques. *IEEE Transactions on Instrumentation and measurement*, 60(11):3592–3607, 2011.

[49] Kenny Davila, Stephanie Ludi, and Richard Zanibbi. Using off-line features and synthetic data for on-line handwritten math symbol recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 323–328. IEEE, 2014.

[50] Stanislas Dehaene. *Consciousness and the brain: Deciphering how the brain codes our thoughts*. Penguin, 2014.

[51] Adrien Delaye and Eric Anquetil. Hbf49 feature set: A first unified baseline for online symbol recognition. *Pattern Recognition*, 46(1):117–130, 2013.

[52] Heather Desurvire, Martin Caplan, and Jozsef A Toth. Using heuristics to evaluate the playability of games. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1509–1512. ACM, 2004.

[53] Anind K Dey. *Providing architectural support for building context-aware applications*. PhD thesis, Atlanta, GA, USA, 2000.

[54] Laslo Dinges, Moftah Elzobi, Ayoub Al-Hamadi, and Zaher Al Aghbari. *Image Processing and Communications Challenges 3*, chapter Synthizing, pages 401–408. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[55] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

[56] Heiko Drewes and Albrecht Schmidt. Interacting with the computer using gaze gestures. In *IFIP Conference on Human-Computer Interaction*, pages 475–488. Springer, 2007.

[57] M-P Dubuisson and Anil K Jain. A modified hausdorff distance for object matching. In *Proceedings of 12th international conference on pattern recognition*, volume 1, pages 566–568. IEEE, 1994.

[58] Krzysztof Duda. Accurate, guaranteed stable, sliding discrete fourier transform [dsp tips & tricks]. *IEEE Signal Processing Magazine*, 27(6):124–127, 2010.

[59] Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, New York, NY, USA, 2000.

[60] David Efron. Gesture and environment. 1941.

[61] Paul Ekman and Wallace V Friesen. The repertoire of nonverbal behavior: Categories, origins, usage, and coding. *semiotica*, 1(1):49–98, 1969.

[62] Randa I Elanwar. The state of the art in handwriting synthesis. In *2nd International Conference on New Paradigms in Electronics & information Technology (peit'013), Luxor, Egypt*, 2013.

[63] Chris Ellis, Syed Zain Masood, Marshall F Tappen, Joseph J Laviola Jr., and Rahul Sukthankar. Exploring the trade-off between accuracy and observational latency in action recognition. *International Journal of Computer Vision*, 101(3):420–436, feb 2013.

[64] Mahmoud Elmezain, Ayoub Al-Hamadi, Jorg Appenrodt, and Bernd Michaelis. A hidden markov model-based continuous gesture recognition system for hand motion trajectory. In *2008 19th International Conference on Pattern Recognition*, pages 1–4. IEEE, 2008.

[65] Hugo Jair Escalante, Isabelle Guyon, Vassilis Athitsos, Pat Jangyodsuk, and Jun Wan. Principal motion components for one-shot gesture recognition. *Pattern Analysis and Applications*, 20(1):167–182, 2017.

[66] Sergio Escalera, Vassilis Athitsos, and Isabelle Guyon. Challenges in multi-modal gesture recognition. In *Gesture Recognition*, pages 1–60. Springer, 2017.

[67] Sergio Escalera, Xavier Baró, Jordi Gonzalez, Miguel A Bautista, Meysam Madadi, Miguel Reyes, Víctor Ponce-López, Hugo J Escalante, Jamie Shotton, and Isabelle Guyon. Chalearn looking at people challenge 2014: Dataset and results. In *European Conference on Computer Vision*, pages 459–473. Springer, 2014.

[68] Faisal Farooq, Damien Jose, and Venu Govindaraju. Phrase-based correction model for improving handwriting recognition accuracies. *Pattern Recogn.*, 42(12):3271–3277, dec 2009.

[69] Anna Maria Feit, Shane Williams, Arturo Toledo, Ann Paradiso, Harish Kulkarni, Shaun Kane, and Meredith Ringel Morris. Toward everyday gaze input: Accuracy and precision of eye tracking and implications for design. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 1118–1130. ACM, 2017.

[70] S Sidney Fels and Geoffrey E Hinton. Glove-talk: A neural network interface between a data-glove and a speech synthesizer. *IEEE transactions on Neural Networks*, 4(1):2–8, 1993.

[71] A Fischer, R Plamondon, C O'Reilly, and Y Savaria. Neuromuscular representation and synthetic generation of handwritten whiteboard notes. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 222–227, sep 2014.

[72] Andreas Fischer, Muriel Visani, Van Cuong Kieu, and Ching Y Suen. Generation of learning samples for historical handwriting recognition using image degradation. In *Proceedings of the 2Nd International Workshop on Historical Document Imaging and Processing*, HIP '13, pages 73–79, New York, NY, USA, 2013. ACM.

[73] Robert Fisher, Jose Santos-Victor, and James Crowley. CAVIAR: Context aware vision using image-based active recognition, 2005.

[74] Norbert Freedman and Stanley P Hoffman. Kinetic behavior in altered clinical states: Approach to objective analysis of motor behavior during clinical interviews. *Perceptual and motor skills*, 24(2):527–539, 1967.

[75] Dustin Freeman, Hrvoje Benko, Meredith Ringel Morris, and Daniel Wigdor. Shadowguides: visualizations for in-situ learning of multi-touch and whole-hand gestures. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 165–172. ACM, 2009.

[76] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[77] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[78] Vittorio Fuccella and Gennaro Costagliola. Unistroke gesture recognition through polyline approximation and alignment. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3351–3354. ACM, 2015.

[79] Hironobu Fujiyoshi, Alan J Lipton, and Takeo Kanade. Real-time human motion analysis by image skeletonization. *IEICE TRANSACTIONS on Information and Systems*, 87(1):113–120, 2004.

[80] Javier Galbally, Julian Fierrez, Marcos Martinez-Diaz, and Javier Ortega-Garcia. Synthetic generation of handwritten signatures based on spectral analysis. In *SPIE Defense, Security, and Sensing*, page 730629. International Society for Optics and Photonics, 2009.

[81] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

[82] B Gatos, T Konidaris, K Ntzios, I Pratikakis, and S J Perantonis. A segmentation-free approach for keyword search in historical typewritten documents. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, ICDAR '05, pages 54–58, Washington, DC, USA, 2005. IEEE Computer Society.

[83] Claude Ghez. The control of movement. In Eric R Kandel, James H, Schwartz, and Thomas M Jessell, editors, *Principles of Neural Science*, chapter 35, pages 533–547. Appleton & Lange, 1991.

[84] Luciano Giromini, Donald J Viglione Jr, Jaime A Pineda, Piero Porcelli, David Hubbard, Alessandro Zennaro, and Franco Cauda. Human movement responses to the rorschach and mirroring activity: An fmri study. *Assessment*, 26(1):56–69, 2019.

[85] David Goldberg and Cate Richardson. Touch-typing with a stylus. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 80–87, New York, NY, USA, 1993. ACM.

[86] Wayne D Gray and Deborah 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(4):322, 2000.

[87] Jason Gregory. *Game Engine Architecture*. Ak Peters Series. Taylor & Francis, 2009.

[88] Sidhant Gupta, Daniel Morris, Shwetak Patel, and Desney Tan. Soundwave: using the doppler effect to sense gestures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1911–1914. ACM, 2012.

[89] T M Ha and H Bunke. Off-line, handwritten numeral recognition by perturbation method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):535–539, may 1997.

[90] Dong Han, Liefeng Bo, and Cristian Sminchisescu. Selection and context for action recognition. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1933–1940, 2009.

[91] Peter Harscher, Rudiger Vahldieck, and Smain Amari. Automated filter tuning using generalized low-pass prototype networks and gradient-based parameter extraction. *IEEE transactions on microwave theory and techniques*, 49(12):2532–2538, 2001.

[92] Sandra G Hart. Nasa-task load index (nasa-tlx); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 50, pages 904–908. Sage publications Sage CA: Los Angeles, CA, 2006.

[93] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[94] Muriel Helmers and Horst Bunke. Generation and use of synthetic training data in cursive handwriting recognition. In *Pattern Recognition and Image Analysis*, pages 336–345. Springer, 2003.

[95] Javier Hernandez-Ortega, Aythami Morales, Julian Fierrez, and Alajandro Acien. Predicting age groups from touch patterns based on neuromotor models. In *International Conference of Pattern Recognition Systems*, pages 1–6. BiDA Lab, 2017.

[96] J Herold and T F Stahovich. The 1¢ 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.

[97] James Herold and Thomas F Stahovich. Speedseg: A technique for segmenting pen strokes using pen speed. *Computers & Graphics*, 35(2):250–264, 2011.

[98] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[99] Michael Hoffman, Paul Varcholik, and Joseph J LaViola Jr. Breaking the status quo: Improving 3d gesture recognition with spatially convenient input devices. In *Virtual Reality Conference (VR), 2010 IEEE*, pages 59–66. IEEE, 2010.

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

[101] Krisztián Horváth and Márton Kuslits. Optimization-based parameter tuning of unscented kalman filter for speed sensorless state estimation of induction machines. In *Electrical and Electronics Engineering (ISEEE), 2017 5th International Symposium on*, pages 1–7. IEEE, 2017.

[102] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[103] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE transactions on information theory*, 8(2):179–187, 1962.

[104] Fuyi Huang, Jiashu Zhang, and Sheng Zhang. Combined-step-size affine projection sign algorithm for robust adaptive filtering in impulsive interference environments. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 63(5):493–497, 2016.

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

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

[107] Nazli Ikizler-Cinbis and Stan Sclaroff. Object, scene and actions: Combining multiple features for human action recognition. In *Computer Vision–ECCV 2010*, pages 494–507. Springer, 2010.

[108] Jana M Iverson and Susan Goldin-Meadow. What's communication got to do with it? gesture in children blind from birth. *Developmental psychology*, 33(3):453, 1997.

[109] Eric Jacobsen and Richard Lyons. The sliding dft. *IEEE Signal Processing Magazine*, 20(2):74–80, 2003.

[110] Eric Jacobsen and Richard Lyons. An update to the sliding dft. *IEEE Signal Processing Magazine*, 21(1):110–111, 2004.

[111] Anil Jain, Karthik Nandakumar, and Arun Ross. Score normalization in multimodal biometric systems. *Pattern recognition*, 38(12):2270–2285, 2005.

[112] Joseph J LaViola Jr. A survey of hand posture and gesture recognition techniques and technology. *Brown University, Providence, RI*, 29, 1999.

[113] Joseph J LaViola Jr and Robert C Zeleznik. Mathpad 2: a system for the creation and exploration of mathematical sketches. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 432–440. ACM, 2004.

[114] Otis M Solomon Jr. Psd computations using welch's method. *NASA STI/Recon Technical Report N*, 92, 1991.

[115] Holger Junker, Oliver Amft, Paul Lukowicz, and Gerhard Tröster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, 2008.

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

[117] Rajesh Kaluri and Ch Pradeep Reddy. A framework for sign gesture recognition using improved genetic algorithm and adaptive filter. *Cogent Engineering*, 3(1):1251730, 2016.

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

[119] Maria Karam and M.c. schraefel. A taxonomy of gestures in human computer interactions. 2005.

[120] Adam Kendon. *Gesture: Visible action as utterance*. Cambridge University Press, 2004.

[121] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting time series: A survey and novel approach. In *Data mining in time series databases*, pages 1–21. World Scientific, 2004.

[122] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[123] Jungsoo Kim, Jiasheng He, Kent Lyons, and Thad Starner. The gesture watch: A wireless contact-free gesture based wrist interface. In *2007 11th IEEE International Symposium on Wearable Computers*, pages 15–22. IEEE, 2007.

[124] Hedvig Kjellström, Javier Romero, and Danica Kragić. Visual object-action recognition: Inferring object affordances from human demonstration. *Comput. Vis. Image Underst.*, 115(1):81–90, jan 2011.

[125] Ming Hsiao Ko, Geoff West, VeSvetha Venkatesh, and Mohan Kumar. Online context recognition in multisensor systems using dynamic time warping. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. Proceedings of the 2005 International Conference on*, pages 283–288, 2005.

[126] Solomon Raju Kota, Jagdish Lal Raheja, Ashutosh Gupta, Archana Rathi, and Shashikant Sharma. Principal component analysis for gesture recognition using systemc. In *2009 International Conference on Advances in Recent Technologies in Communication and Computing*, pages 732–737. IEEE, 2009.

[127] Suleyman Serdar Kozat, Alper Tunga Erdogan, Andrew C Singer, and Ali H Sayed. Steady-state mse performance analysis of mixture approaches to adaptive filtering. *IEEE Transactions on Signal Processing*, 58(8):4050–4063, 2010.

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

[129] Sven Kratz and Michael Rohs. Protractor3d: a closed-form solution to rotation-invariant 3d gestures. In *Proceedings of the 16th international conference on Intelligent User Interfaces*, pages 371–374. ACM, 2011.

[130] Per Ola Kristensson, Thomas Nicholson, and Aaron Quigley. Continuous recognition of one-handed and two-handed gestures using 3d full-body motion tracking sensors. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pages 89–92. ACM, 2012.

[131] Eyal Krupka, Kfir Karmon, Noam Bloom, Daniel Freedman, Ilya Gurvich, Aviv Hurvitz, Ido Leichter, Yoni Smolin, Yuval Tzairi, Alon Vinnikov, and Others. Toward realistic hands gesture interface: Keeping it simple for developers and machines. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 1887–1898. ACM, 2017.

[132] Arun Kulshreshth, Chris Zorn, and Joseph J LaViola. Poster: Real-time markerless kinect based finger tracking and hand gesture recognition for hci. In *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 187–188. IEEE, 2013.

[133] Gord Kurtenbach and Eric A Hulteen. Gestures in human-computer communications. In Brenda Laurel, editor, *The Art of Computer-Human Interface Design*, pages 309—-317. Addison-Wesley, 1990.

[134] Michael H Kutner, Christopher J Nachtsheim, John Neter, and William Li. *Applied linear statistical models*, volume 5. McGraw-Hill Irwin New York, 2005.

[135] Doo Young Kwon. *A design framework for 3D spatial gesture interfaces*. PhD thesis, 2008.

[136] Joseph J LaViola. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proceedings of the workshop on Virtual environments 2003*, pages 199–206. ACM, 2003.

[137] Joseph J LaViola and Joseph J LaViola Jr. 3d gestural interaction: The state of the field. *ISRN Artificial Intelligence*, 2013, 2013.

[138] Joseph J LaViola Jr. and Robert C Zeleznik. A practical approach for writer-dependent symbol recognition using a writer-independent symbol recognizer. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(11):1917–1926, 2007.

[139] Do-Hoon Lee and Hwan-Gue Cho. A new synthesizing method for handwriting korean scripts. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(01):45–61, 1998.

[140] Ju-Won Lee and Gun-Ki Lee. Design of an adaptive filter with a dynamic structure for ecg signal processing. *International Journal of Control, Automation, and Systems*, 3(1):137–142, 2005.

[141] Luis A Leiva, Daniel Martín-Albo, and Réjean Plamondon. Gestures à go go: Authoring synthetic human-like stroke gestures using the kinematic theory of rapid movements. *ACM Trans. Intell. Syst. Technol.*, 7(2):15:1—-15:29, nov 2015.

[142] Luis A Leiva, Daniel Martín-Albo, and Radu-Daniel Vatavu. Synthesizing stroke gestures across user populations: A case for users with visual impairments. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 4182–4193. ACM, 2017.

[143] Yang 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.

[144] Hui Liang, Junsong Yuan, Daniel Thalmann, and Nadia Magnenat Thalmann. Ar in hand: Egocentric palm pose tracking and gesture recognition for augmented reality applications. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 743–744. ACM, 2015.

[145] Laurence Likforman-Sulem, Anna Esposito, Marcos Faundez-Zanuy, Stéphan Clémençon, and Gennaro Cordasco. Emothaw: A novel database for emotional state recognition from handwriting and drawing. In *IEEE TRANSACTIONS ON HUMAN-MACHINE SYSTEMS, VOL. 47*, pages 273–284. IEEE, 2017.

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

[147] David Llorens, Federico Prat, Andrés Marzal, Juan Miguel Vilar, María José Castro, Juan-Carlos Amengual, Sergio Barrachina, Antonio Castellanos, Salvador España Boquera, J A Gómez, and Others. The ujipenchars database: a pen-based database of isolated handwritten characters. In Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, and Daniel Tapias, editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may 2008. European Language Resources Association (ELRA).

[148] Hao Lü, James A Fogarty, and Yang Li. Gesture script: Recognizing gestures and their structure using rendering scripts and interactively trained parts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1685–1694, New York, NY, USA, 2014. ACM.

[149] Emilie Lundin, Håkan Kvarnström, and Erland Jonsson. A synthetic fraud data generation methodology. In *Proceedings of the 4th International Conference on Information and Communications Security*, ICICS '02, pages 265–277, London, UK, UK, 2002. Springer-Verlag.

[150] Gil Luria and Sara Rosenblum. Comparing the handwriting behaviours of true and false writing with computerized handwriting measures. In *Applied Cognitive Psychology, issue 24*, pages 1115–1128. Department of Human Services, Haifa University, 2010.

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

[152] Granit Luzhnica, Jorg Simon, Elisabeth Lex, and Viktoria Pammer. A sliding window approach to natural hand gesture recognition using a custom data glove. In *3D User Interfaces (3DUI), 2016 IEEE Symposium on*, pages 81–90. IEEE, 2016.

[153] Scott MacLean, David Tausky, George Labahn, Edward Lank, and Mirette Marzouk. Tools for the efficient generation of hand-drawn corpora based on context-free grammars. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 125–132, New York, NY, USA, 2009. ACM.

[154] Robert Macrae and Simon Dixon. Accurate real-time windowed time warping. In *in ISMIR, 2010*, pages 423–428, 2010.

[155] Mehran Maghoumi and Joseph J LaViola Jr. Deepgru: Deep gesture recognition utility. *CoRR*, abs/1810.1, 2018.

[156] Nathan Magrofuoco, Paolo Roselli, JL Pérez Medina, Jean Vanderdonckt, and Santiago Villarreal. Vector-based, structure preserving stroke gesture recognition. *Training*, 4(q3):q4, 2019.

[157] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. Hand gesture recognition with leap motion and kinect devices. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 1565–1569. IEEE, 2014.

[158] Nicolai Marquardt, Ricardo Jota, Saul Greenberg, and Joaquim A Jorge. The continuous interaction space: interaction techniques unifying touch and gesture on and above a digital surface. In *IFIP Conference on Human-Computer Interaction*, pages 461–476. Springer, 2011.

[159] Kyle A Martin and Joseph J Laviola. The transreality interaction platform: Enabling interaction across physical and virtual reality. In *Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2016 IEEE International Conference on*, pages 177–186. IEEE, 2016.

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

[161] Daniel Martín-Albo, Réjean Plamondon, and Enrique Vidal. Training of on-line handwriting text recognizers with synthetic text generated using the kinematic theory of rapid human movements. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 543–548. IEEE, 2014.

[162] David McNeill. *Hand and mind: What gestures reveal about thought*. University of Chicago press, 1992.

[163] Melanie Mitchell and John H Holland. When will a genetic algorithm outperform hill-climbing? 1993.

[164] Thomas M Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[165] Darnell J Moore, Irfan A Essa, and Monson H Hayes III. Exploiting human actions and object context for recognition tasks. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 80–86 vol.1, 1999.

[166] Louis-Philippe Morency and Trevor Darrell. Head gesture recognition in intelligent interfaces: The role of context in improving recognition. In *Proceedings of the 11th international conference on Intelligent user interfaces*, pages 32–38. ACM, 2006.

[167] Meredith Ringel Morris, Jacob O Wobbrock, and Andrew D Wilson. Understanding users' preferences for surface gestures. In *Proceedings of graphics interface 2010*, pages 261–268. Canadian Information Processing Society, 2010.

[168] Arya Chowdhury Mugdha, Ferdousi Sabera Rawnaque, and Mosabber Uddin Ahmed. A study of recursive least squares (rls) adaptive filter algorithm in noise removal from ecg signals. In *Informatics, Electronics & Vision (ICIEV), 2015 International Conference on*, pages 1–6. IEEE, 2015.

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

[170] Shrikanth Narayanan and Panayiotis G Georgiou. Behavioral signal processing: Deriving human behavioral informatics from speech and language. *Proceedings of the IEEE*, 101(5):1203–1233, 2013.

[171] R Navaratnam, A W Fitzgibbon, and R Cipolla. The joint manifold model for semi-supervised multi-valued regression. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, oct 2007.

[172] Pedro Neto, Dário Pereira, J Norberto Pires, and A Paulo Moreira. Real-time and continuous hand gesture spotting: an approach based on artificial neural networks. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 178–183. IEEE, 2013.

[173] Natalia Neverova, Christian Wolf, Graham W Taylor, and Florian Nebout. Multi-scale deep learning for gesture detection and localization. In *European Conference on Computer Vision*, pages 474–490. Springer, 2014.

[174] Diederick C Niehorster, Li Li, and Markus Lappe. The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research. *i-Perception*, 8(3):2041669517708205, 2017.

[175] Yuan Niu and Hao Chen. Gesture authentication with touch input for mobile devices. In *International Conference on Security and Privacy in Mobile Information and Communication Systems*, pages 13–24. Springer, 2011.

[176] Juliet Norton, Chadwick A Wingrave, and Joseph J. LaViola Jr. Exploring strategies and guidelines for developing full body video game interfaces. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 155–162. ACM, 2010.

[177] Uran Oh and Leah Findlater. The challenges and potential of end-user gesture customization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1129–1138. ACM, 2013.

[178] Kenton O'Hara, Abigail Sellen, Gerardo Gonzalez, Tom Carrel, Helena Mentis, Graeme Penney, Antonio Criminisi, Robert Corish, Mark Rouncefield, Neville Dastur, and Andreas Varnavas. Touchless interaction in surgery. *Communications of the ACM*, 57(1):70–77, 2014.

[179] Ryuichi Oka. Spotting method for classification of real world data. *The Computer Journal*, 41(8):559–565, 1998.

[180] Christopher Olah. Understanding lstm networks, 2015.

[181] Karen Otte, Bastian Kayser, Sebastian Mansow-Model, Julius Verrel, Friedemann Paul, Alexander U Brandt, and Tanja Schmitz-Hübsch. Accuracy and reliability of the kinect version 2 for clinical measurement of motor function. *PloS one*, 11(11):e0166532, 2016.

[182] Maja Pantic, Alex Pentland, Anton Nijholt, and Thomas S Huang. Human computing and machine understanding of human behavior: A survey. In *Artifical Intelligence for Human Computing*, pages 47–71. Springer, 2007.

[183] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, 36(2):3336–3341, 2009.

[184] Rachel Patel, Beryl Plimmer, John Grundy, and Ross Ihaka. Ink features for diagram recognition. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, pages 131–138. ACM, 2007.

[185] Orasa Patsadu, Chakarida Nukoolkit, and Bunthit Watanapa. Human gesture recognition using kinect camera. In *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE)*, pages 28–32. IEEE, 2012.

[186] Vladimir I Pavlovic, Rajeev Sharma, and Thomas S Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):677–695, 1997.

[187] Andriy Pavlovych and Wolfgang Stuerzlinger. The tradeoff between spatial jitter and latency in pointing tasks. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, pages 187–196. ACM, 2009.

[188] Ken Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pages 287–296, New York, NY, USA, 1985. ACM.

[189] David Pinelle, Nelson Wong, and Tadeusz Stach. Heuristic evaluation for games: usability principles for video game design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1453–1462. ACM, 2008.

[190] Corey Pittman, Pamela Wisniewski, Conner Brooks, and Joseph J LaViola Jr. Multiwave: Doppler effect based gesture recognition in multiple dimensions. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1729–1736. ACM, 2016.

[191] Corey R. Pittman, Eugene M. Taranta II, and Joseph J. LaViola Jr. A $-family friendly approach to prototype selection. In *International Conference on Intelligent User Interfaces, Proceedings IUI*, volume 07-10-Marc, 2016.

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

[193] Réjean Plamondon. A kinematic theory of rapid human movements. part i: Movement representation and control. *Biological Cybernetics*, 72(4):309–320, mar 1995.

[194] Réjean Plamondon. A kinematic theory of rapid human movements. part ii. movement time and control. *Biological cybernetics*, 72 4:309–20, 1995.

[195] Réjean Plamondon and Moussa Djioua. A multi-level representation paradigm for handwriting stroke generation. *Human movement science*, 25(4-5):586–607, 2006.

[196] Rejean Plamondon, Chunhua Feng, and Anna Woch. A kinematic theory of rapid human movement. part iv: a formal mathematical proof and new insights. *Biological Cybernetics*, 89(2):126–138, 2003.

[197] Stergios Poularakis, Grigorios Tsagkatakis, Panagiotis Tsakalides, and Ioannis Katsavounidis. Sparse representations for hand gesture recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3746–3750. IEEE, 2013.

[198] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes in C*, volume 2. Cambridge University Press, New York, NY, USA, 1996.

[199] Francis K H Quek. Toward a vision-based hand gesture interface. In *Virtual reality software and technology*, pages 17–31. World Scientific, 1994.

[200] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[201] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

[202] Vilayanur S Ramachandran. *The tell-tale brain: A neuroscientist's quest for what makes us human*. WW Norton & Company, 2012.

[203] Vilayanur S Ramachandran, Sandra Blakeslee, and Neil Shah. *Phantoms in the brain: Probing the mysteries of the human mind*. William Morrow New York, 1998.

[204] Vilayanur S Ramachandran and Edward M Hubbard. Synaesthesia–a window into perception, thought and language. *Journal of consciousness studies*, 8(12):3–34, 2001.

[205] Michalis Raptis, Darko Kirovski, and Hugues Hoppe. Real-time classification of dance gestures from skeleton animation. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 147–156. ACM, 2011.

[206] Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*, 2004.

[207] Chotirat Ann Ratanamahatana and Eamonn Keogh. Three myths about dynamic time warping data mining. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 506–510. SIAM, 2005.

[208] 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.

[209] Kishore K Reddy and Mubarak Shah. Recognizing 50 human action categories of web videos. *Machine Vision and Applications*, pages 1–11, 2012.

[210] Jun Rekimoto. Gesturewrist and gesturepad: Unobtrusive wearable interaction devices. In *Wearable Computers, 2001. Proceedings. Fifth International Symposium on*, pages 21–27. IEEE, 2001.

[211] Jose A Rodriguez-Serrano and Florent Perronnin. Synthesizing queries for handwritten word image retrieval. *Pattern Recognition*, 45(9):3270–3276, 2012.

[212] H A Rowley, M Goyal, and J Bennett. The effect of large training set sizes on online japanese kanji and english cursive recognizers. In *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*, pages 36–40, 2002.

[213] Wenjie Ruan, Quan Z Sheng, Lei Yang, Tao Gu, Peipei Xu, and Longfei Shangguan. Audiogest: enabling fine-grained hand gesture detection by decoding echo signal. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 474–485. ACM, 2016.

[214] Dean Rubine. *The automatic recognition of gestures*. PhD thesis, Citeseer, 1991.

[215] Dean Rubine. Specifying gestures by example. *SIGGRAPH Computer Graphics*, 25(4):329–337, jul 1991.

[216] Fabrizio Russo. Technique for image denoising based on adaptive piecewise linear filters and automatic parameter tuning. *IEEE transactions on instrumentation and measurement*, 55(4):1362–1367, 2006.

[217] Suman Kr Saha, Soumya Sarkar, Rajib Kar, Durbadal Mandal, and S P Ghoshal. Digital stable iir low pass filter optimization using particle swarm optimization with improved inertia weight. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pages 147–152. IEEE, 2012.

[218] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[219] Yasushi Sakurai, Christos Faloutsos, and Masashi Yamamuro. Stream monitoring under the time warping distance. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 1046–1055. IEEE, 2007.

[220] Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture recognition with a wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14. ACM, 2008.

[221] Jeremy Scott, David Dearman, Koji Yatani, and Khai N Truong. Sensing foot gestures from the pocket. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 199–208. ACM, 2010.

[222] Junjie Shan and Srinivas Akella. 3d human action segmentation and recognition using pose kinetic energy. In *Advanced Robotics and its Social Impacts (ARSO), 2014 IEEE Workshop on*, pages 69–75. IEEE, 2014.

[223] J Shotton, A Fitzgibbon, M Cook, T Sharp, M Finocchio, R Moore, A Kipman, and A Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 1297–1304, Washington, DC, USA, 2011. IEEE Computer Society.

[224] David B Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning Proceedings 1994*, pages 293–301. Elsevier, 1994.

[225] Steven W Smith. *The scientist and engineer's guide to digital signal processing*. California Technical Pub. San Diego, 1997.

[226] Sang-Wook Sohn, Young-Bin Lim, Jae-Jun Yun, Hun Choi, and Hyeon-Deok Bae. A filter bank and a self-tuning adaptive filter for the harmonic and interharmonic estimation in power signals. *IEEE Transactions on Instrumentation and Measurement*, 61(1):64–73, 2012.

[227] Min-Ho Song and Rolf Inge Godøy. How fast is your body motion? determining a sufficient frame rate for an optical motion tracking system using passive markers. *PloS one*, 11(3):e0150993, 2016.

[228] Yale Song, David Demirdjian, and Randall Davis. Continuous body and hand gesture recognition for natural human-computer interaction. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2(1):5, 2012.

[229] Petre Stoica and Randolph L Moses. *Spectral analysis of signals*, volume 1. Pearson Prentice Hall Upper Saddle River, NJ, 2005.

[230] Eugene M. Taranta II, Seng Lee Koh, Brian M. Williamson, Kevin P. Pfeil, Corey R. Pittman, and Joseph J. LaViola Jr. Pitch pipe: An automatic low-pass filter calibration technique for pointing tasks. In *Proceedings of Graphics Interface 2019*, GI 2019. Canadian Information Processing Society, 2019.

[231] Eugene M Taranta II and Joseph J LaViola Jr. Penny pincher: A blazing fast, highly accurate $-family recognizer. In *Proceedings of the 41st Graphics Interface Conference*, volume 2015-June of *GI '15*, pages 195–202, Toronto, Ont., Canada, Canada, 2015. Canadian Information Processing Society.

[232] Eugene M Taranta II, Mehran Maghoumi, Corey R Pittman, and Joseph J LaViola Jr. 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, pages 873–885, New York, NY, USA, 2016. ACM, ACM.

[233] Eugene M. Taranta II, Corey R. Pittman, Mehran Maghoumi, Mykola Maslych, Yasmine Moolenaar, and Joseph J. LaViola Jr. Machete: Easy, efficient, and precise continuous custom gesture segmentation. In *(To Appear) ACM Transactions on Computer-Human Interaction (TOCHI)*, 2020.

[234] Eugene M. Taranta II, Corey R. Pittman, Jack P. Oakley, Mykola Maslych, Mehran Maghoumi, and Joseph J. LaViola Jr. 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*, pages 1–11, 2020.

[235] Eugene M Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R Pittman, and Joseph J LaViola Jr. 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, pages 5850–5861, New York, NY, USA, 2017. ACM.

[236] Eugene M Taranta II, Thaddeus K Simons, Rahul Sukthankar, and Joseph J Laviola Jr. Exploring the benefits of context in 3d gesture recognition for game-based virtual environments. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(1):1, 2015.

[237] Eugene M. Taranta II, A.N. Vargas, S.P. Compton, and J.J. Laviola. A dynamic pen-based interface for writing and editing complex mathematical expressions with math boxes. *ACM Transactions on Interactive Intelligent Systems*, 6(2), 2016.

[238] Eugene M Taranta II, Andrés N Vargas, and Joseph J LaViola Jr. Streamlined and accurate gesture recognition with penny pincher. *Computers & Graphics*, 55:130–142, 2016.

[239] Dana Tenneson. Chempad: A pedagogical tool for exploring handwritten organic molecules. Technical report, Technical Report, No. RI 02912, Brown University, 2005.

[240] Achint Oommen Thomas, Amalia Rusu, and Venu Govindaraju. Synthetic handwritten captchas. *Pattern Recogn.*, 42(12):3365–3373, dec 2009.

[241] Emanuel Trabes and Mario A Jordan. Self-tuning of a sunlight-deflickering filter for moving scenes underwater. In *Information Processing and Control (RPIC), 2015 XVI Workshop on*, pages 1–6. IEEE, 2015.

[242] Tamás Varga and Horst Bunke. Generation of synthetic training data for an hmm-based handwriting recognition system. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 618–622 vol.1, aug 2003.

[243] Tamás Varga and Horst Bunke. Offline handwriting recognition using synthetic training data produced by means of a geometrical distortion model. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(07):1285–1302, 2004.

[244] Tamás Varga, Daniel Kilchhofer, and Horst Bunke. Template-based synthetic handwriting generation for the training of recognition systems. In *Proceedings of the 12th Conference of the International Graphonomics Society*, pages 206–211, 2005.

[245] Radu-Daniel Vatavu. 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, pages 271–278, New York, NY, USA, 2011. ACM.

[246] Radu-Daniel 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.

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

[248] Radu-Daniel Vatavu. 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, pages 4667–4679, New York, NY, USA, 2017. ACM.

[249] Radu-Daniel Vatavu, Lisa Anthony, and Quincy Brown. Child or adult? inferring smartphone users' age group from touch measurements alone. In *IFIP Conference on Human-Computer Interaction*, pages 1–9. Springer, 2015.

[250] Radu-Daniel Vatavu, Lisa Anthony, and Jacob 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.

[251] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. Relative accuracy measures for stroke gestures. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, ICMI '13, pages 279–286, New York, NY, USA, 2013. ACM, ACM.

[252] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. Gesture heatmaps: Understanding gesture performance with colorful visualizations. In *Proceedings of the 16th International Conference on Multimodal Interaction*, pages 172–179. ACM, 2014.

[253] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. $ 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*, page 23. ACM, 2018.

[254] Radu-Daniel Vatavu, Laurent Grisoni, and Stefan-Gheorghe Pentiuc. Multiscale detection of gesture patterns in continuous motion trajectories. In *International Gesture Workshop*, pages 85–97. Springer, 2009.

[255] Radu-Daniel Vatavu and Stefan-Gheorghe Pentiuc. Multi-level representation of gesture as command for human computer interaction. *Computing and Informatics*, 27(6):837–851, 2012.

[256] Radu-Daniel Vatavu, Stefan-Gheorghe Pentiuc, Laurent Grisoni, Christophe Chaillou, and Stefan cel Mare. Modeling shapes for pattern recognition: A simple low-cost spline-based approach. *Advances in Electrical and Computer Engineering*, 8(15):67–71, 2008.

[257] Radu-Daniel Vatavu, Daniel Vogel, Géry Casiez, and Laurent 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.

[258] Ondrej Velek and Masaki Nakagawa. *Document Analysis Systems V: 5th International Workshop, DAS 2002 Princeton, NJ, USA, August 19–21, 2002 Proceedings*, chapter The Impact, pages 106–110. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[259] Duc-Hoang Vo, Huu-Hung Huynh, Thanh-Nghia Nguyen, and Jean Meunier. Automatic hand gesture segmentation for recognition of vietnamese sign language. In *Proceedings of the Seventh Symposium on Information and Communication Technology*, pages 368–373. ACM, 2016.

[260] Tian-Shu Wang, Heung-Yeung Shum, Ying-Qing Xu, and Nan-Ning Zheng. Unsupervised analysis of human gestures. In *Pacific-Rim Conference on Multimedia*, pages 174–181. Springer, 2001.

[261] Daniel Weinland, Remi Ronfard, and Edmond Boyer. A survey of vision-based methods for action representation, segmentation and recognition. *Comput. Vis. Image Underst.*, 115(2):224–241, feb 2011.

[262] Gregory F Welch. History: The use of the kalman filter for human motion tracking in virtual reality. *Presence: Teleoperators and Virtual Environments*, 18(1):72–91, 2009.

[263] Peter Welch. The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. *IEEE Transactions on audio and electroacoustics*, 15(2):70–73, 1967.

[264] Kevin R Wheeler. Device control using gestures sensed from emg. In *Soft Computing in Industrial Applications, 2003. SMCia/03. Proceedings of the 2003 IEEE International Workshop on*, pages 21–26. IEEE, 2003.

[265] Jacob O Wobbrock, Leah Findlater, Darren Gergle, and James J Higgins. 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, pages 143–146, New York, NY, USA, 2011. ACM.

[266] Jacob O Wobbrock, Meredith Ringel Morris, and Andrew D Wilson. User-defined gestures for surface computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1083–1092. ACM, 2009.

[267] Jacob O Wobbrock, Andrew D Wilson, and Yang 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.

[268] Di Wu, Fan Zhu, and Ling Shao. One shot learning gesture recognition from rgbd images. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 7–12. IEEE, 2012.

[269] Jianxin Wu, Adebola Osuntogun, Tanzeem Choudhury, Matthai Philipose, and James M Rehg. A scalable approach to activity recognition based on object use. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[270] Robert Xiao, Chris Harrison, Karl D D Willis, Ivan Poupyrev, and Scott E Hudson. Lumitrack: low cost, high precision, high speed tracking with projected m-sequences. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 3–12. ACM, 2013.

[271] Hee-Deok Yang, Stan Sclaroff, and Seong-Whan Lee. Sign language spotting with a threshold model based on conditional random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(7):1264–1277, 2009.

[272] Yina Ye and Petteri Nurmi. Gestimator: Shape and stroke similarity based gesture recognition. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 219–226. ACM, 2015.

[273] Ying Yin and Randall Davis. Gesture spotting and recognition using salience detection and concatenated hidden markov models. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 489–494. ACM, 2013.

[274] Hwanjo Yu and Sungchul Kim. Svm tutorial—classification, regression and ranking. *Handbook of Natural computing*, pages 479–506, 2012.

[275] Robert C Zeleznik, Kenneth P Herndon, and John F Hughes. Sketch: An interface for sketching 3d scenes. In *ACM SIGGRAPH 2007 courses*, page 19. ACM, 2007.

[276] Hansong Zeng and Yi Zhao. Sensing movement: Microsensors for body motion measurement. *Sensors*, 11(1):638–660, 2011.

[277] Yaodong Zhang and James R Glass. An inner-product lower-bound estimate for dynamic time warping. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5660–5663. IEEE, 2011.

[278] Yougen Zhang, Wei Deng, Hanchen Song, and Lingda Wu. A fast pen gesture matching method based on nonlinear embedding. In Tieniu Tan, Qiuqi Ruan, Xilin Chen, Huimin Ma, and Liang Wang, editors, *Advances in Image and Graphics Technologies*, volume 363 of *Communications in Computer and Information Science*, pages 223–231. Springer Berlin Heidelberg, 2013.

[279] Xin Zhao, Xue Li, Chaoyi Pang, Quan Z Sheng, Sen Wang, and Mao Ye. Structured streaming skeleton–a new feature for online human gesture recognition. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(1s):22, 2014.

[280] Zhiwei Zhu, Vlad Branzoi, Mikhail Sizintsev, Nicholas Vitovitch, Taragay Oskiper, Ryan Villamil, Ali Chaudhry, Supun Samarasekera, and Rakesh Kumar. Ar-weapon: live augmented reality based first-person shooting system. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 618–625. IEEE, 2015.