Unmediated Interaction: Communicating with Computers and Embedded Devices as If They Are Not There

Brian A. Smith

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2018

ABSTRACT

Unmediated Interaction: Communicating with Computers
and Embedded Devices as If They Are Not There
Brian A. Smith

Although computers are smaller and more readily accessible today than they have ever been, I believe that we have barely scratched the surface of what computers can become. When we use computing devices today, we end up spending a lot of our time navigating to particular functions or commands to use devices *their* way rather than executing those commands immediately.

In this dissertation, I explore what I call *unmediated interaction*, the notion of people using computers as if the computers are not there and as if the people are using their own abilities or powers instead. I argue that facilitating unmediated interaction via personalization, new input modalities, and improved text entry can reduce both *input overhead* and *output overhead*, which are the burden of providing inputs to and receiving outputs from the intermediate device, respectively.

I introduce three computational methods for reducing input overhead and one for reducing output overhead. First, I show how *input data mining* can eliminate the need for user inputs altogether. Specifically, I develop a method for mining controller inputs to gain deep insights about a players playing style, their preferences, and the nature of video games that they are playing, all of which can be used to personalize their experience without any explicit input on their part. Next, I introduce *gaze locking*, a method for sensing eye contact from an image that allows people to interact with computers, devices, and other objects just by looking at them. Third, I introduce computationally optimized keyboard designs

for touchscreen manual input that allow people to type on smartphones faster and with far fewer errors than currently possible. Last, I introduce the *racing auditory display (RAD)*, an audio system that makes it possible for people who are blind to play the same types of racing games that sighted players can play, and with a similar speed and sense of control as sighted players. The RAD shows how we can reduce output overhead to provide user interface parity between people with and without disabilities.

Together, I hope that these systems open the door to even more efforts in unmediated interaction, with the goal of making computers less like devices that we use and more like abilities or powers that we have.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank several people who have supported me in my graduate studies and who I owe much of my success as a researcher to. I feel very fortunate to have had the mentors, role models, collaborators, and friends that I did.

Thank you to my advisor Shree Nayar for introducing me to the world of research, for showing me what great research is all about, and for supporting me as I grew and continue to grow as an inventor, teacher, and scholar. Shree took a chance on me when I had no research experience, gave me a long leash from which to explore ideas (and sometimes fail at them), and conveyed trust in my ability to cross the finish line that I at times did not have in myself. He guided me through the twists and turns of my evolving interests even as they strayed outside of his usual research areas, and likewise helped me navigate many of the twists and turns of my life in general. I hope to become half as inspiring and impactful to others as you have been to me and to so many. I also hope to see you at a conference someday.

Thank you to my co-advisor Steve Feiner for welcoming me to his lab and treating me as one of its members when I was most doubtful about my research and needed his outreach the most. Steve spent countless hours — sometimes past midnight — listening to my ideas, exposing me to new aspects of human–computer interaction research, and

helping me refine my ideas in ways that I could confidently call my own. Though Steve's knowledge of the field is almost otherworldly, he is always grounded in the trenches of his students' work. Thank you, Steve, for creating a wonderful research environment for all who visit your lab.

Thank you to my other dissertation committee members, John Kender, Lydia Chilton, and Barbara Tversky, for serving on the committee and for their feedback on my manuscript and presentation. John paid meticulous detail to my manuscript and many of my arguments, Lydia was instrumental in helping me position my dissertation work in the broader context of human–computer interaction, and Barbara raised very interesting questions pertaining to cognition that I am considering exploring in future research.

Thank you to Julia Hirschberg for taking me in as an NSF Integrated Graduate Education and Research Traineeship (IGERT) Trainee. Much of my data science background comes from the From Data to Solutions IGERT program she directed, without which Chapter 3 of this dissertation would not be possible. The program also exposed me to fields such as market structure research, participatory sensing, and bioinformatics that are very much relevant to human–computer interaction.

Thank you to Konstantin Voevodski, Xiaojun Bi, and Shumin Zhai for hosting me during my two internships at Google (one with Konstantin and one with Xiaojun and Shumin). They showed me how to approach research in cases when it can affect millions of people, and Xiaojun and Shumin were collaborators of mine for the project described in Chapter 5. I regret not being able to work with them more.

Thank you to my many friends and colleagues at Columbia's Vision and Graphics Center: in particular, Computer Vision Laboratory (CAVE) members, Computer Graph-

ics and User Interfaces (CGUI) Laboratory members, and Visual Appearance Laboratory (VAPLAB) members. Thank you to Daniel Sims for help conducting user studies and preparing figures related to the racing auditory display (Chapter 6).

Thank you to so many current and former faculty members and administrators at Columbia's Department of Computer Science, including Anne Fleming, Augustin Chaintreau, Changxi Zheng, Eugene Wu, Dave Blei, Janet Kayfetz, Daisy Nguyen, Remi Moss, Jessica Rosa, Twinkle Edwards, Cindy Walters, and Elias Tesfaye, all of whom contributed to my success as a researcher and my general well-being as a graduate student.

Thank you to all of my user study participants, and special thanks to Russell Martello and Gus Chalkias at Helen Keller Services for the Blind (HKSB) for help recruiting participants for the racing auditory display's user study, which is featured in Chapter 6.

Thank you to Sean Pagaduan for helping me become a better writer and communicate my research more effectively. Thank you, as well, for being my de-facto editor and for helping me frame my research in interesting new ways. And thank you, most of all, for believing in me, for believing in my work, and for supporting me during the difficult years of my graduate studies.

Finally, I thank my father for being with me every step of the way and every step to come.

To my father Rudolph G. Smith

Chapter 1

Introduction

Although the last few decades have brought computers much closer to becoming small, readily accessible devices through which to perform tasks, I believe that we have barely scratched the surface of what computers can become. Their evolution so far — from mainframes that only specialists could use to desktop personal computers (PCs), laptops, smartphones, and new form factors such as smartwatches and head-worn displays — has been described as a movement toward increased *computational intimacy* [Harrison 2013] since computers are becoming closer and closer to people's bodies and lives, the benefit being that computers are becoming much smaller and easier for the average person to use. Still, computers have many of the same problems today as they did decades ago.

Almost every device lives in its own ecosystem, for example, separate from other systems and with its own unique interface for controlling it, requiring users to switch between many different controllers and interfaces when using these devices. Smaller devices are difficult to control and type on, and users must configure each device or system to suit their preferences individually — a painstaking process. In general, we end up spending a lot of our time on computers navigating to particular functions or commands rather than executing those commands. Computers themselves may be getting more capable, but that capability does not always extend to us.

I believe that this problem stems from how we have grown to view computers over the last few decades: as being intermediaries between people and the tasks they would like to perform. People must often focus their attention on computers and use them *their* way to perform their tasks successfully.

In this dissertation, I will explore what I have come to call *unmediated interaction*, an interaction mentality that I believe can reduce or eliminate the burden of using computers. Unmediated interaction is the notion of people using computers as if the computers are not there and as if the people are using their own abilities or powers instead. An example would be the ability to turn on a lamp or other device just by looking at it and perhaps making a quick hand gesture instead of interacting with a physical switch as an intermediary. Devices that facilitate unmediated interaction are ones that work to minimize the perception of themselves as intermediaries between people and the computing tasks those people would like to perform.

The idea that computers should strive to become transparent to the user is not new. Rutkowski, for example, describes "the ideal relationship between user and tool" as *transparency*, in which "[t]he user is able to apply intellect directly to the task; the tool itself seems to disappear" [Rutkowski 1982]. Shneiderman describes *direct manipulation* as a system model that can turn users' "grudging acceptance or outright hostility" toward using an interactive system into "glowing enthusiasm" and delight [Shneiderman 1983]. It is the notion of a system "display[ing] a representation of the objects of interest [to the user] and permit[ting] rapid, incremental, reversible operations through physical actions rather than command syntax" [Shneiderman 1983] — in other words, allowing a system's users to perform tasks by directly manipulating graphical representations of objects in the software.

Hutchins et al., using Laurel's concept of *first-personness* [Laurel 1986] (published subsequently) as a lens, identifies *direct engagement* as being an important aspect of direct manipulation [Hutchins, Hollan, and Norman 1985]. Specifically, it is that of "provid[ing] the user with a world in which to interact" so that "objects [. . .] behave as if they are the real thing, [. . .] remov[ing] the perception of the computer as an intermediary" [Hutchins, Hollan, and Norman 1985]. As Shneiderman himself explains, "[t]he trick in creating a direct manipulation system is to come up with an appropriate representation or model of reality" [Shneiderman 1983]. Such thinking embodies the approach that we have taken to create these types of interfaces so far.

The difference between unmediated interaction as I am defining it and these other concepts — with direct engagement being the closest in meaning — is that unmediated interaction frames the concept of direct engagement in terms of today's world of embedded computing and the Internet of Things (IoT), in which the objects of interest to the user are no longer mere representations of real-world objects but rather objects and devices themselves. The choice of how to represent these objects is largely moot because they already exist in the real world; our focus for achieving unmediated interaction will therefore be to reduce the overhead associated with the computing intermediary being present. I use the term *unmediated* instead of *direct* for the same reason: when the object of interest and the intermediate device are one and the same, users can be said to interact with the object *directly* even if there is great overhead in doing so, whereas the interaction can only be described as *unmediated* when the overhead is unnoticeably small. I believe that just as direct engagement (and by extension, direct manipulation) in software can improve users' interaction speed, effectiveness, and sense of satisfaction as Shneiderman holds [Shneider-

man 1983], unmediated interaction with today's embedded computing devices can improve interaction speed, effectiveness, and users's sense of satisfaction when using them.

1.1 The Makings of Unmediated Interaction

The point of unmediated interaction is to reduce the burden of using a device as an intermediary for performing a computing task as much as possible, ideally completely. The burden occurs twice: when users provide inputs to a device (i.e., command the device), and when users interpret outputs from the device. These moments roughly correspond to Norman's gulfs of execution and evaluation, respectively [Norman 2013]. Hence, in order to make interacting with computers and devices unmediated, we must explore ways of reducing both *input overhead* and *output overhead* as I will call them.

To reduce output overhead, a device should display its output immediately, continuously, and in a manner that is representative of the device's current state. The output should also be easy for the user to understand at a glance, without having to "parse" or translate it in order to draw insights from it. These maxims might seem obvious but in certain contexts — particularly in cases in which devices cannot have standard displays such as screens or must accommodate users with disabilities such as blindness — fulfilling them is a rich research challenge.

Devices should be able reduce input overhead in three ways. First, they should be able to eliminate the need for users to provide them with inputs whenever possible by anticipating what that input would be. Doing so requires learning a personalized model of users' habits, needs, and predicted behavior. A smart thermostat, for example, should be

able to learn a user's preferred temperature schedule so that it can adjust the temperature on its own without requiring anymore of the user's input.

Second, for cases in which anticipating the users' inputs is not possible but those inputs are basic, the device should be able to offer an input modality such as gaze that is near instantaneous. Going back to the smart thermostat, it should allow users who would like to deviate from their habitual temperature schedule to indicate so and change the temperature instantly — without the overhead of walking to the thermostat and navigating its menus — by perhaps looking at it from across the room and making a quick hand gesture.

Finally, for cases in which anticipating the users' inputs is not possible and those inputs are complex, the device should offer users as painless a typing experience as possible. For example, suppose that the smart thermostat allowed users to give their preset temperature schedules names. In this case, naming the schedule requires complex input that cannot be entered with just a glance, so the thermostat should make the process of typing as easy as possible, with an on-body keyboard perhaps or at the very least a touchscreen (soft) keyboard.

1.2 Research Questions

Each of the requirements for facilitating unmediated interactions that I just described is the basis for a research question that I explore in this dissertation. The first three questions relate to reducing input overhead and the last relates to reducing output overhead.

RQ 1. *How can we help eliminate the need for users to enter inputs on devices completely?*

Specifically, how can we personalize a person's experience with a computer or a device without requiring them to input their preferences explicitly?

Personalization is key to computers being able to understand and anticipate what users will want in different contexts. It is also the most useful way for devices to offer unmediated interaction because it can preclude users from having to enter inputs altogether. Recommendation systems are becoming a fact of life and are present in services ranging from Amazon to Netflix. The key challenge is to personalize a user's experience or recommend experiences for the user based on experiences that they have already had, without requiring any explicit input from the user.

As an example, we will soon live in a world in which people can shop among many different types of virtual reality (VR) experiences on demand in pay-per-view or Netflix-type marketplaces. These experiences could include going on tours, going to the arcade, playing escape room puzzles, and visiting theme park attractions. When that happens, finding experiences that a person would like based on the history of experiences that they liked before will be difficult. We will explore how to do this in the context of video games — a domain in which we can track what users do completely — using nothing more than the controller inputs that they used to play the game anyway.

RQ 2. *In cases where it is not possible to anticipate users' inputs but the inputs themselves are basic, how can we make inputting nearly instantaneous?*

Often, the only input we need to give a device is a simple command such as turning the device on or off or asking the device to display the previous or next item in a list. For these types of interactions, navigating to the particular device that we would like to command

is the most time-consuming part. Light switches, remote controls, and settings menus in apps and programs are just a few examples. If computers and devices could sense attention themselves so that we could interact with them by looking at them, using them would be much less cumbersome.

A promising and very natural way of enabling computers and devices to sense attention is to incorporate gaze tracking systems into them. The idea has not gained much ground, however, because gaze tracking systems suffer from many limitations: they only work at close range (80 cm or less), require each user to undergo a calibration process, are not robust to varying head poses, and they are active, meaning they require special infrared illumination hardware. As a result, I explore — along with my colleagues — whether there is a better way for computers to sense attention, and work to develop a technique for doing so that does not suffer from these limitations.

RQ 3. *In cases where it is not possible to anticipate users' inputs and the inputs themselves are complex, how can we make typing — specifically on small devices — as painless as possible? Specifically, how can we make typing on small devices such as smartphones as fast and as accurate as typing on large devices such as desktop PCs?*

Perhaps the central usability problem pertaining to smartphones, smartwatches, and other small devices is that of entering text. While button presses and touchscreen gestures for scrolling, zooming, and switching applications can get users far in their interactions with those devices, typing is inevitable. Touchscreen typing in particular is slow and prone to errors and typos.

Gesture typing, the concept of drawing word gestures on a touchscreen by swiping

to connect words' letters, has been proven to be faster than touch typing but suffers from a major drawback not present in touch typing: word gesture ambiguity. Many words such as "or" and "our" have identical gestures on the Qwerty keyboard, and many more such as "pretty" and "prey" have very similar gestures, all because users must swipe over unintended letters to reach intended ones. We will explore how we can modify existing gesture typing keyboards to make typing much faster and more error-free.

RQ 4. *How can we reduce output overhead to accommodate users with disabilities? Specifically, how can we make it possible for people who are blind to play video games — a real-time visual interactive system — without slowing down the interaction?*

As mentioned earlier, people spend much of their time using today's devices on just navigating to particular functions or commands rather than executing those commands. The situation is bad already but is even worse for people with disabilities, who must often use specialized interfaces or devices that are much slower and more cumbersome to use than what others can use.

Providing user interface parity between people with and without disabilities is challenging; user interface designers lose some or all of the channels for providing inputs to or getting outputs from a computer system. Designers lose some of the expressiveness or "throughput" that they can achieve via a visual display when designing for users who are visually impaired, for example, and lose that type of display entirely for users who are blind. Likewise, they lose input throughput when designing for users with motor impairments. As a result, I will explore how to achieve as similar a throughput as possible using a challenging domain — making racing games equally accessible to people who are blind

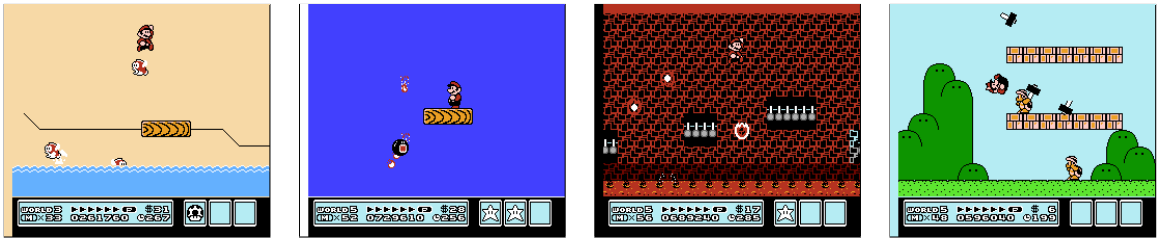
— as a case study.

1.3 Contributions

This dissertation contributes to human–computer interaction in three primary ways. First, we describe the notion of *unmediated interaction*, which is when computing devices allow users to perform tasks with them as if the devices themselves are not there and as if the user is using their own abilities or powers instead. Second, we argue that devices must reduce both input overhead and output overhead to near-zero levels to facilitate unmediated interaction. In particular, input overhead can be reduced by eliminating the need for users to enter inputs altogether, by making the process of entering basic inputs nearly instantaneous, and by making typing complex inputs as painless as possible.

Last, and most significantly, we explore each of the aforementioned methods for reducing input overhead — personalization for eliminating the need to provide inputs, new input modalities for making basic input nearly instantaneous, and less laborious typing for complex input — and do so in a computational manner. We do so by performing *input data mining* on users' inputs to achieve personalization, by developing a new computer vision approach for sensing eye contact to achieve near-instantaneous input, and by optimizing touchscreen keyboards for gesture typing to make typing less laborious. We also contribute a system for reducing output overhead by providing a computational representation of a video game's current state to blind players.

I summarize each contribution in detail below. Each of them brings us closer to the goal of truly unmediated interaction, making computers less like devices that we use and



(a) World 3-2 (b) World 5-9 (c) World 5-Fortress2 (d) Hammer Bros.

Figure 1.1: Stage recommendations from controller inputs. Although these stages look quite different at first glance, they all share the same primary type of gameplay: jumping around flying enemies or projectiles. Here we see (a) Flying Cheep-Cheeps, (b) Fire Chomps and their fireballs, (c) Podoboos, and (d) Hammer Bros. *Super Mario Bros. 3* © Nintendo.

more like abilities or powers that we have.

C 1. *A method for mining controller inputs to recognize individual video game players and personalize the experiences for them:*

Here, I show how to make it possible for a system to find experiences that a person would like based on ones that they liked before by analyzing the content of those experiences: what having those experiences entails. This problem is common in the realm of video games, which I use as a domain. Super Mario Maker for the Wii U, for example, features a marketplace with hundreds of thousands of player-created levels that are not curated in any way, making it difficult for players to find levels that they might enjoy playing. The method that I contribute, which is to perform data mining on *input words* that are formed from players' controller inputs, can be used as the basis for a Netflix-type recommendation system for video games and virtual reality experiences.

The method itself involves observing a player's controller inputs (raw actions) as they play a video game level and using advanced statistical inference techniques to infer

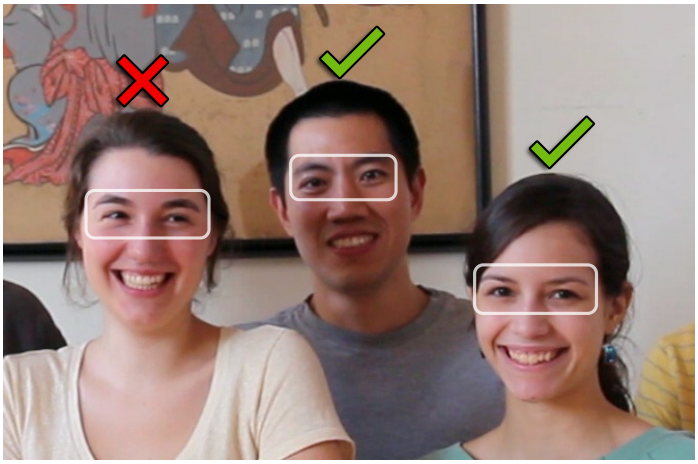
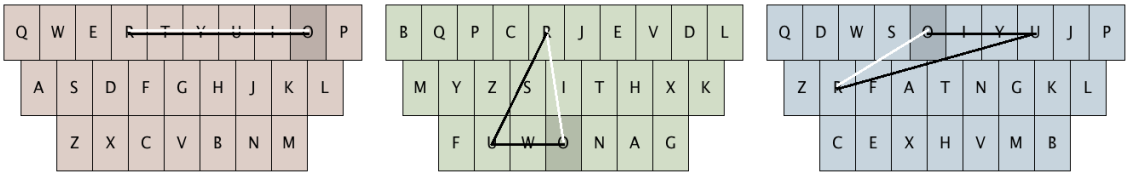


Figure 1.2: Gaze locking. We propose the idea of sensing eye contact directly from an image in a passive, appearance-based manner. The main idea is to focus on gaze *locking* (a binary problem) rather than gaze tracking (a continuous problem) and exploit the special appearance of direct eye gaze. Our approach can be used to facilitate a wide range of applications.

information about both the game level and the player themselves from those inputs. Regarding the game level, the system infers the types of action that it fosters, such as puzzle solving and jumping on narrow platforms, and uses that information to recommend levels that play similarly. Figure 1.1 shows an example of this. Regarding the player themselves, the system can learn their unique playing style and subsequently recognize them in just 20 seconds of gameplay. Neither form of understanding is obvious to a human observer, but the system can nonetheless infer these from the nuances of players' controller input behaviors.

C 2. *A gaze "locking" system for interacting with objects just by looking at them:*

First, I propose the idea of simplifying the continuous gaze tracking problem into a binary gaze "locking" problem: that is, detecting eye contact instead of exact gaze direction. I then developed a computer vision system that can sense attention by detecting eye contact



(a) Qwerty (b) GK-D (c) GK-T

Figure 1.3: Keyboards optimized for gesture typing. The 'o' key is shaded to mark the beginning of the word gestures for "or" (white) and "our" (black). (a) The Qwerty keyboard suffers from the problem of gesture ambiguity. Many pairs of words (such as "or" and "our" shown here) share the same gesture on Qwerty. (b) The GK-D keyboard ("Gesture Keyboard—Double optimized") is the best compromise for gesture clarity and gesture speed. Here, the gestures for "or" and "our" are noticeably different. (c) The GK-T keyboard ("Gesture Keyboard—Triple optimized") is the best compromise for gesture clarity, gesture speed, and Qwerty similarity. Again, the two gestures are noticeably different.

directly from images and video. It exploits the special appearance of direct eye gaze, which is a subtle difference from slightly averted gaze. The resulting system, shown in Figure 1.2, is calibration-free, requires no extra hardware, and is over 90% accurate at detecting eye contact from a distance of 18 meters.

The idea came from noticing the difference between how people seem to sense attention compared to how gaze tracking systems sense attention. People have difficulty determining the angle that someone else is looking at but seem to be very good at determining when someone else is looking *at them*. Moreover, most important interactions between people involve determining whether the other person is looking at them.

C 3. *A touchscreen keyboard optimized for gesture typing:*

With this contribution, I show how to modify existing gesture typing keyboards to make typing on small devices much faster and more error-free than is currently possible. Specifically, I explore modifying Qwerty, the standard keyboard layout, to make word gestures

shorter and more distinct without making users have to learn how to type all over again.

To see if a touchscreen keyboard layout can be changed in a beneficial and easy-to-learn way, I developed three models for predicting a given keyboard layout's worth. The models were: (1) gesture clarity, which models how distinct a keyboard layout's word gestures are; (2) gesture speed, which models how quickly users can type on a given keyboard layout based on human motor control theory; and (3) learnability, which models how easy a given keyboard layout would be to learn. By performing a rigorous optimization procedure using these models, I found that error rates can be reduced by 52% over Qwerty. Figure 1.3 shows two keyboards optimized for gesture typing.

C 4. *The racing auditory display (RAD):*

The racing auditory display, or the RAD for short, is an audio system makes it possible for people who are blind to play the same types of racing games as sighted players can, with a similar speed and sense of control as what sighted players have. It works with a standard pair of headphones and comprises two novel sonification techniques: the *sound slider* for understanding a car's speed and trajectory on a racetrack and the *turn indicator system* for alerting players to the direction, sharpness, length, and timing of upcoming turns. Figure 1.4 illustrates the RAD.

The RAD shows how computation and clever system design can combine to allow users with disabilities — in this case, users who are blind — to control computers with a throughput very similar to users without disabilities. The RAD's focus is on achieving output (display) parity compared to what sighted players receive, and it does so by presenting players who are blind with stimuli that allows them to make the same moment-to-moment

Figure 1.4: Playing racing games without sight. A study participant who is congenitally blind playing a racing game uses the racing auditory display (RAD), which outputs spatialized sound through a standard pair of headphones. Using the RAD, players can understand their car's pose, their car's speed, and the direction, sharpness, length, and timing of upcoming turns.

decisions that sighted players make while they race.

More specifically, the RAD distills many pieces of information — the car's lateral position on the track, its heading with respect to the track's, its speed, the track's width, whether the track is about to immediately turn, and more — into a single measure that is no less relevant to the process of racing than all of that information put together. Moreover, it does so in a way that gives players the freedom to decide how riskily they would like to race: whether they should cut corners by racing close to the track's inside edge or stay safe by racing closer to the track's center. I liken this process of distilling the many pieces of information to a more compact, salient form to that of dimensionality reduction in machine learning and statistics.

Chapter 2

Related Work

Given the varied nature of this dissertation work, this section is organized around each contribution. The work on mining controller inputs to understand gameplay and personalize games for players builds on game analytics systems (Section 2.1) as well as work in probabilistic topic modeling (Section 2.2). The gaze locking system builds on work in gaze estimation and tracking (Section 2.3), gaze perception (Section 2.4), and gaze-based interactive systems (Section 2.5). The touchscreen keyboard optimized for gesture typing builds on existing stroke-based virtual keyboards (Section 2.6) and keyboard layout optimization work (Section 2.7). The racing auditory display builds on earlier audio navigation systems (Section 2.8) and earlier blind-accessible racing games and driver assistance systems (Section 2.9).

2.1 Game Analytics

Game analytics research is focused on developing new ways to extract meaningful information about players and games. This information includes understanding the players themselves, what is happening in the game, and to what extent players are enjoying the game. Together, this information allows game developers to mold game content to players' tastes, perform matchmaking in multiplayer games, discover game exploits and illicit ac-

tivity, and refine their games to create a desired experience for the player. El-Nasr et al.'s recent book [El-Nasr, Drachen, and Canossa 2013] provides a comprehensive review of the field.

To understand the players themselves, game analytics systems compute metrics from logs of user-initiated events (UIEs), which record information about what is happening in the game such as items used, enemies defeated, and causes of death. Such an approach is inspired by how Hurst et al. [Hurst, Hudson, and Mankoff 2007] determine a software user's skill level from metrics such as the depth of the user's menu selections and the average velocity of the user's mouse cursor during those selections. Likewise, Buckley et al. [Buckley, Chen, and Knowles 2013] use mouse- and keyboard-based metrics to determine players' skill levels. From the UIE-based metrics, the game analytics systems can predict when players will stop playing [Mahlmann, Drachen, Togelius, Canossa, and Yannakakis 2010]; cluster players based on skill level; and cluster players into behavioral personas such as pacifists, puzzle solvers, and assassins [Drachen, Canossa, and Yannakakis 2009; Drachen, Sifa, Bauckhage, and Thurau 2012; Gow, Baumgarten, Cairns, Colton, and Miller 2012]. They can also discover illicit behavior such as game bots [Kang, Woo, Park, and Kim 2013] and real money trading (RMT) [Itsuki, Takeuchi, Fujita, and Matsubara 2010] by finding outliers from the clustering.

To understand how players are experiencing a game, game analytics systems combine UIE logs with attitudinal data from surveys and, sometimes, with gameplay videos [Kim, Gunn, Schuh, Phillips, Pagulayan, and Wixon 2008] or affect measures [Yannakakis and Togelius 2011]. Together, the data form records of which parts of a game work well and which parts of a game do not. From these records, developers can learn which game param-

eters need to be tuned [Kim, Gunn, Schuh, Phillips, Pagulayan, and Wixon 2008; Pedersen, Togelius, and Yannakakis 2009; Shaker, Yannakakis, and Togelius 2011] and can even tune them on the fly [Hastings and Stanley 2010; Yannakakis and Hallam 2009; Yannakakis and Togelius 2011].

Our key observation is that today's game analytics systems can only reveal information about games that is expressible in terms of UIEs. If we depict video games as Moore finite state machines (FSMs) [Moore 1956] as shown in Figure 3.1, we can see that this type of information can only be found at the edge circled in purple in that figure. Other types of information — such as the nuances of each player's control styles, the types of action that a game fosters, and the extent that each game level fosters each type of action — remain out of reach of today's systems. By observing the player's controller inputs — the edge circled in green in Figure 3.1 — we can begin to understand and describe these other types of information in a quantitative way.

2.2 Probabilistic Topic Modeling

Probabilistic topic modeling is concerned with developing unsupervised algorithms for discovering the topics or themes present in a corpus of documents. Latent Dirichlet allocation (LDA) [Blei, Ng, and Jordan 2003] is the simplest topic model and was first used to analyze text corpora such as news articles and scientific papers. Over the last decade, LDA has been used to infer topics from source code [Lukins, Kraft, and Etzkorn 2008; Maskeri, Sarkar, and Heafield 2008], genes [Pritchard, Stephens, and Donnelly 2000], speech and audio [Chien and Chueh 2008; Kim, Narayanan, and Sundaram 2009], and images [Blei

and Jordan 2003; Sivic, Russell, Zisserman, Freeman, and Efros 2008]. It has also been used to detect spam and fraud [Bíró, Siklósi, Szabó, and Benczúr 2009; Xing and Girolami 2007]. Blei [Blei 2012] provides an excellent overview of topic modeling and its applications.

The present work is, to the best of our knowledge, the first to extend topic modeling to the realms of user inputs and gameplay. In doing so, we solve two challenges. The first challenge is mapping input streams to discrete words that represent gameplay primitives and are compatible with probabilistic topic models; we solve this with our concept of *input words* (Section 3.1). The second challenge is isolating the effect of a player's play style from our understanding of the gameplay types present in a game; we solve this by proposing the *player–gameplay action (PGA) model* (Section 3.3), a novel extension of LDA.

2.3 Gaze Estimation and Tracking

Both gaze estimation and gaze tracking (gaze estimation at video rate) have been studied extensively in the past few decades. Hansen and Ji [Hansen and Ji 2010] and Morimoto and Mimica [Morimoto and Mimica 2005] provide excellent surveys of earlier work. Ideally, a gaze tracking system should be accurate, passive, non-intrusive, calibration-free, and robust to distance and head pose [Morimoto and Mimica 2005]. Unfortunately, current systems maintain accuracy at the expense of other qualities—they are predominantly active systems that work only at close range (80 cm or less).

For example, Morimoto et al.'s [Morimoto, Amir, and Flickner 2002] and Beymer and

Flickner's [Beymer and Flickner 2003] feature-based techniques are accurate and robust to head pose, but are active, require calibration, and work only at close range. Baluja and Pomerleau's [Baluja and Pomerleau 1994] and Tan et al.'s [Tan, Kriegman, and Ahuja 2002] appearance-based techniques are accurate and calibration-free, but are active, sensitive to head pose, and work only at close range. Hansen and Pece's [Hansen and Pece 2005] commercial off-the-shelf (COTS) system is passive and easy to calibrate, but is sensitive to head pose and again works only at close range. Nishino and Nayar's method [Nishino and Nayar 2004] is passive and produces an image of what a person is looking at, but requires a high resolution image of the eye to provide useful results. Stiefelhagen et al.'s eye tracker [Stiefelhagen, Yang, and Waibel 1997] is passive and boasts an accuracy of up to 1.3°, but requires a fixed head pose and was only tested on four users at close range. They also developed an earlier gaze tracking system [Stiefelhagen, Yang, and Waibel 1996] that tracks only head pose and not eye gaze direction. The gaze locking approach that we develop in Chapter 4, by contrast, can be applied to any image (including those from COTS products or even from the Web) and is accurate, passive, non-intrusive, calibration-free, and robust to distance and head pose.

2.4 Gaze Perception

A number of studies evaluating people's perception of gaze have been performed, starting with Gibson and Pick's study [Gibson and Pick 1963] using six subjects and an in-person gazer. It concludes that, at 2 m, people sense eye contact when others are looking between the left and right edges of their face, but the authors urge a repeat of the experiment using

a model (e.g., a set of photographs) as stimulus instead of a live person. Cline's in-person study [Cline 1967] using a half-mirror incorporates several head poses and eye occlusion, confirming Gibson and Pick's eye contact results and measuring errors in the perception of gaze direction in general. Gamer and Hecht [Gamer and Hecht 2007] explore the effects of distance, eye occlusion, and the presence of a second head as well, while Martin and Jones [Martin and Jones 1982] examine the effects of distance and lighting intensity from a signal detection standpoint. Symons et al. [Symons, Lee, Cedrone, and Nishimura 2004] focus on triadic eye gaze acuity (the ability to judge where someone is looking in space) rather than dyadic eye gaze acuity, and verify that digital photographs are a good substitute for in-person gazers. Gemmell et al. [Gemmell, Toyama, Zitnick, Kang, and Seitz 2000] and Chen [Chen 2002] explore how the design of videoconferencing systems can promote gaze awareness without using special-purpose hardware.

2.5 Gaze-Based Interactive Systems

The pursuit of more natural, ubiquitous user interfaces has been an important goal for the HCI community. A new class of user interfaces, called *attentive* user interfaces, aims to facilitate more social interactions between users and devices by treating users' attention as a valuable resource [Vertegaal 2003]. In doing so, they must (a) sense users' attention, (b) make inferences about what users want to do, and (c) negotiate "turns" amongst themselves, as Vertegaal and Shell describe [Vertegaal and Shell 2008].

To date, however, these interfaces have been limited by current gaze tracking techniques, making it difficult to sense users' attention. Although sensing attention through eye

contact alone would be ideal, many systems incorporate either gesture-based control [Bolt 1980], manual input [Zhai, Morimoto, and Ihde 1999], or intrusive head-mounted cameras [Smith, Vertegaal, and Sohn 2005] as a workaround. Shell et al., however, did in fact propose standalone eye contact sensors [Shell, Vertegaal, Cheng, Skaburskis, Sohn, Stewart, Aoudeh, and Dickie 2004], but they use active infrared illumination and interfere with each other if placed within 80° of visual angle of each other. They extend Morimoto et al.'s PupilCam design [Morimoto, Koons, Amir, and Flickner 2000] (which locates pupils by reflecting infrared light on them) by comparing the location of a corneal glint (i.e., the first Purkinje image) with that of the pupil reflection. The gaze locking technique that we develop in Chapter 4, by contrast, is completely passive, is not prone to interference, and is accurate at long range. We compare the two techniques directly in Section 4.4.

Omron's commercial OKAO Vision system [Omron 2012] includes a passive gaze tracker, and Ye et al. [Ye, Li, Fathi, Han, Rozga, Abowd, and Rehg 2012] combine this with an active and intrusive head-mounted camera in order to determine mutual gaze (simultaneous eye contact) between the person wearing the camera and another person. However, Ye at al. find OKAO Vision's gaze tracker to be inaccurate and sensitive to head pose. Their resulting system has an MCC of 0.72, was only tested on one pair of subjects, and was not shown to work over long distances. Our gaze locking approach is accurate, passive, non-intrusive, robust to head pose, and achieves an MCC of over 0.83 at distances of 18 m.

2.6 Stroke-Based Virtual Keyboards

Cirrin [Mankoff and Abowd 1998] and Quikwriting [Perlin 1998] are the first virtual keyboards designed specifically for word-level unistroke text entry. In both of those keyboards, users trace gestures that alternate from the center of a radial layout to one or more zones around the center (representing characters), with one articulation per character. However, since these gestures are completely determined by character layout without statistical pattern recognition, the letter layout has to be one-dimensional and most word-gestures defined on these layouts are very complex.

SHARK [Zhai and Kristensson 2003] and SHARK^2 [Kristensson and Zhai 2004] introduced gesture typing as we know it today. Gesture typing is also known as shape writing and the word-gesture keyboard paradigm. In these systems, users gesture words by swiping from letter to letter on a virtual Qwerty keyboard. The word gestures are much simpler than they are on Cirrin and Quikwriting, but since Qwerty has no central "dead zone" for strokes to cross from character to character, users must stroke over unintended characters to reach intended ones, causing an inherent ambiguity in word gestures compared to Qwerty (see Figure 5.1(a)). Even with sophisticated models for predicting users' intended words, Bi et al. [Bi, Chelba, Ouyang, Partridge, and Zhai 2012] found that the error rate from gesture typing is 5–10% higher than that from touch typing.

2.7 Keyboard Layout Optimization

As has been widely published [Rick 2010; Yamada 1980; Zhai, Hunter, and Smith 2002], Qwerty was designed to reduce jamming in mechanical typewriters by placing common

digraphs (consecutive letter pairs) on opposite sides of the keyboard. Though this works well for two-handed or two-finger typing, researchers have long acknowledged that this is unsuitable for one-finger typing [Getschow, Rosen, and Goodenough-Trepagnier 1986; Lewis, Kennedy, and LaLomia 1999]. There have been many proposed optimized keyboard layouts over the years for both bimanual [Oulasvirta, Reichel, Li, Zhang, Bachynskyi, Vertanen, and Kristensson 2013] and unimanual typing [Bi, Smith, and Zhai 2012; Bi, Smith, and Zhai 2010; Dunlop and Levine 2012; MacKensie and Zhang 1999; Rick 2010; Zhai, Hunter, and Smith 2000; Zhai, Hunter, and Smith 2002]. Most of these layouts were optimized for touch typing, but the Square OSK layout [Rick 2010] was optimized for stroking.

We should emphasize, however, that existing optimized layouts are predominantly optimized for typing speed — essentially minimizing finger travel distance — and that optimizing for word gesture clarity is an entirely different, and often conflicting, problem. As an example, the Dvorak layout arranges common letters in the home row to make bimanual typing faster [Yamada 1980], but this also makes word gestures more similar (and less unique, hurting gesture clarity) since many paths between keys become straight lines on the home row. In Chapter 5 we optimize for word gesture clarity along with gesture typing speed and Qwerty similarity.

As another example, the ATOMIK [Zhai, Hunter, and Smith 2002] and Square ATOMIK [Zhai and Kristensson 2010] keyboards were optimized for speed with a bias for having keys appear in alphabetical order. Although these keyboards were tuned so that the gestures for 17 common words were short and memorable, they were not specifically optimized for gesture clarity. These keyboards, in fact, predate gesture typing altogether.

Other examples include Quasi-Qwerty [Bi, Smith, and Zhai 2010], which was optimized for speed and familiarity, and the Sath keyboards [Dunlop and Levine 2012], which were optimized for those metrics plus tap interpretation clarity for improved spell checking.

Few optimized layouts have gained widespread adoption. This is likely due to both learnability and the complexity of tapping input: users may type with one, two, or even ten fingers, and a good layout must accommodate each. The increasing popularity of gesture typing, however, may offer a better chance at introducing new layouts since most users gesture words with one finger and our optimized layouts significantly improve both accuracy and speed over Qwerty.

2.8 Audio Navigation Systems

Audio navigation systems help people who are blind navigate on foot from one place to another in the real world. They consist of a GPS tracker, a computing device, and a pair of headphones. Perhaps the most archetypal examples are audioGPS [Holland, Morse, and Gedenryd 2002] and SWAN [Wilson, Walker, Lindsay, Cambias, and Dellaert 2007] (short for System for Wearable Audio Navigation), which both guide users from their current location to their destination via a sequence of waypoints that they must reach along the way. The user must follow a sound known as an *acoustic beacon* to travel from waypoint to waypoint until they reach their destination.

Most research in this area has focused on how to perfect these types of systems, such as discovering which type of sounds are easier to localize and follow [Tran, Letowski, and Abouchacra 2000] or how large each waypoint's "capture radius" should be [Walker

and Lindsay 2004; Walker and Lindsay 2006]. These systems, however, are unsuitable for racing games for two reasons. First, they assume that the user is walking and has the flexibility to stop and rotate to center the acoustic beacon in front of them. Second, using these systems amounts to simply following orders, while video games should afford players a high sense of control over what they are doing.

2.9 Blind-Accessible Racing Games

A number of driving systems and racing games currently employ mechanisms to assist drivers and players who are blind. Here, we will survey three blind-accessible video games and a blind driver assistance system, each employing a different user interface for driving a car on a virtual track. We can only show screenshots of the latter two systems because the first two do not have graphics.

Blindfold Racer (iOS, 2014)

Blindfold Racer [Shultz 2014a] is an audio racing game developed on iOS by Marty Schultz as part of his series of blind-accessible smartphone games. In *Blindfold Racer*, players steer by rotating their mobile device left and right as they would a steering wheel. The goal is to drive to the end of a track without hitting fences on the track's sides. The player can also adjust their speed to three fixed values by swiping up or down on their device's touchscreen. The game outputs sound in stereo and pans a music track between the left and right channels as a means of displaying the car's lateral position on the track. It will play exclusively in the left channel if the player's car is adjacent to the left side of the track and

vice-versa.

Treasure and animals are indicated using repeating audio samples that grow louder as the player approaches them. The player should try to center the sounds of treasure between the left and right channels to collect the treasure, and they should keep the sounds of animals panned to the left or right to avoid hitting the animals.

With respect to Figure 6.2, we would classify *Blindfold Racer* as an efficiency-preserving game. While *Blindfold Racer* moves at a pace that is just as fast as racing games with graphics, the three elements of player intention as described in the previous section are limited in *Blindfold Racer* compared to racing games that sighted players would play.

In *Blindfold Racer*, it is not possible for the player to anticipate upcoming turns, accelerate and decelerate in an analog manner, or perform higher level strategies such as cutting corners. In fact, the developer explains that there is no concept of vehicle physics, that tracks are modeled using a simplified geometry that requires straightaways to be in the same direction and all turns to be less than 90° [Shultz 2013], and that car steering is simplified so that the car will move in that straightaway direction when the mobile device is tilted to the center position [Shultz 2014b].

Mach 1 (PC, 2003)

Mach 1 [audiogames archive 2015] is an audio-based racing game published on PC by Jim Kitchen eleven years before *Blindfold Racer* was released. The player's goal is similar that from *Blindfold Racer*, but in *Mach 1* there are no obstacles present on the track. Players accelerate, decelerate, and steer using a USB steering wheel or controller joystick, and they

can press a button to have a voice speak their current lateral position on the track: a number from 1 to 100 where 1 represents the track's left edge, 100 the right edge, and 50 the center. The player should tap the button repeatedly to monitor their lateral position continuously.

As the player approaches an upcoming turn, the game will loop a predetermined sound effect in the left or right stereo audio channel depending on the direction of the turn. The sound effect starts playing quietly but grows louder as the player approaches the beginning of the turn. The game will play a thumping sound as the player reaches the turn, and the process will repeat to signify the end of the turn: a random looping sound effect growing louder followed by a thump.

Unlike *Blindfold Racer*, *Mach 1* allows players to anticipate upcoming turns and accelerate and decelerate in an analog manner. Still, players do not have full freedom to "read the road" since it is difficult to determine from the increasing volume effect exactly when a turn will begin and the game only alerts players of a single upcoming turn or straightaway at a time. As with *Blindfold Racer*, there is no concept of vehicle physics, tracks are modeled using a simplified geometry (so there is no concept of cutting corners), and car steering is simplified so that the car will move in a straightaway direction whenever the player lets go of the steering.

Top Speed Series (PC, 2004)

The *Top Speed* [Ruijter, Ruijter, Duvigneau, and Loots 2004] series is a series of racing games released on PC by a team of four developers. The goal for players is the same as in *Blindfold Racer* and *Mach 1*: to reach the end of the track as quickly as possible without

hitting the sides. *Top Speed 2* and *3* support multiplayer races, though the cars cannot collide with each other or interact with each other in any way. Players steer with a joystick controller as in *Mach 1*.

Like *Blindfold Racer*, a sound is panned between the left and right channels as a means of displaying the player's lateral position on the track. In the *Top Speed* series, however, that sound is the sound of the player's car's engine. A speech clip saying a phrase such as "easy left" or "hard right" will play when the player enters a turn. These phrases describe the direction and sharpness of the turn, and the player must react quickly by steering the appropriate amount. As with *Blindfold Racer* and *Mach 1*, there is no concept of vehicle physics, tracks are modeled using a simplified geometry, and car steering is simplified so that the car will move in a straightaway direction when the player lets go of the steering.

Sucu and Folmer's Haptic Steering Interface

Sucu and Folmer's haptic steering interface [Sucu and Folmer 2014] is a driver assistance system published as a response to the National Federation of the Blind's Blind Driver Challenge [National Federation of the Blind 2013], an initiative to make it possible for people who are blind to drive a car by themselves. The driver steers with a steering wheel and has rumble motors (in this case, PlayStation Move controllers) attached to the back of their hands.

At each time step, the system computes the location of what Sucu and Folmer call a *target point*, which is the point on the median of the track a fixed distance ahead of the driver's current position. If the car's current heading points too far away from the target

point's direction, the system will vibrate the left or right rumble motors. The analogy is that of a rumble strip on the side of a highway: if the vibration is felt on the right the player should steer to the left and vice-versa.

Although the authors state that making a racing game with this system is promising future work, we feel that its current goal as a driver assistance system runs contrary to supporting intention. When drivers use this system, they must follow its orders as soon as those orders are felt and nothing more. Our interface, by contrast, is designed to support player intention.

Chapter 3

Mining Controller Inputs to Understand Gameplay

In many ways, personalization is key to helping computers help people. By understanding each person's preferences and abilities and adapting to them, computers should be able to give each person experiences that feel tailored to them individually, making each person feel as if the world is designed for them.

In this section, we explore whether it is possible to personalize a user's experience or recommend experiences for the user based on experiences that they have already had, without requiring any explicit input from the user. More specifically, we develop a system that can analyze the content of experiences — what having those experiences entails — to find new experiences that are similar. We can think of this as categorizing experiences and in an automatic way.

We explore how to do this in the context of video games since we can track what a user does completely in this domain using nothing more than the controller inputs that they used to play the game anyway. The resulting system can be used as the basis for a Netflix-type recommendation system for video games and virtual reality experiences as well as a way for game developers to analyze the experiences that players are having in their games.

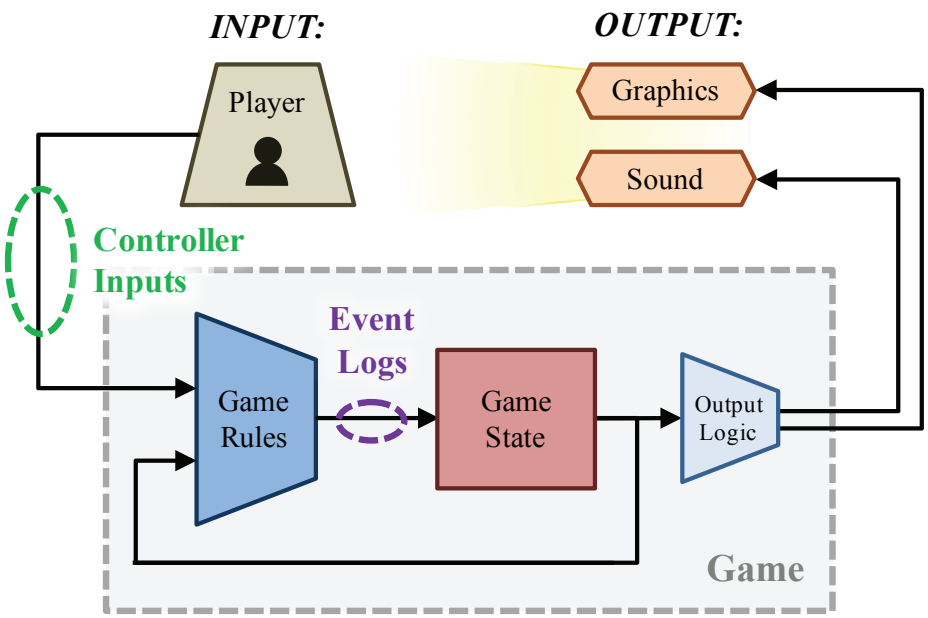


Figure 3.1: Games as Moore finite state machines [Moore 1956]. Each node and edge can be observed to learn about different aspects of how players are experiencing a game. Game analytics as we know it today is based on event logs triggered by state transitions: the edge circled in purple. We show how to use the player's raw controller inputs — the edge circled in green — to understand and describe gameplay in a quantitative way. Our methods can complement those based on event logs and those based on the game's output to paint a fuller picture of the experiences that players have with games.

Controller Inputs as a Gameplay Signal

Today's game analytics systems use event logging code to help developers understand how players are experiencing their games. By recording many aspects of what is happening in games — items used, enemies defeated, causes of death, and much more — event logs can help developers discover game sections that are too difficult [Kim, Gunn, Schuh, Phillips, Pagulayan, and Wixon 2008], cluster players into meaningful types [Drachen, Canossa, and Yannakakis 2009; Drachen, Sifa, Bauckhage, and Thurau 2012; Gow, Baumgarten, Cairns, Colton, and Miller 2012], predict when players will stop playing [Mahlmann, Drachen, Togelius, Canossa, and Yannakakis 2010], and more to varying degrees of success. They cannot, however, capture other information about gameplay that may be of interest to de-

velopers — information such as whether there is enough variety in the action from level to level and whether particular game levels feature the appropriate style of gameplay. Indeed, the concept of gameplay itself is difficult to define and quantify.

Here, we will show how to use a very different source of information — the player's raw controller inputs — to understand and describe gameplay in a quantitative way. The key idea is to form *input words* from the stream of controller inputs to represent *gameplay primitives*: sequences of inputs occurring within small periods of time. We can then treat the problem of describing the types of gameplay — or action — in a game as a case of probabilistic topic modeling [Blei 2012], in which the "topics" are gameplay types, the "documents" are play sessions for sections of the game, and the "words" are input words.

We show that just as topic models such as latent Dirichlet allocation (LDA) [Blei, Ng, and Jordan 2003] can discover the prevailing topics in a corpus of news articles — topics such as science and politics — they can discover the main types of action in a video game when applied to streams of controller inputs. In *Super Mario Bros. 3* [Nintendo 1988], a classic platforming game that we use as a case study, making precise running jumps onto narrow platforms and jumping around flying enemies and projectiles are two such gameplay types. We also show that just as topic models output each article's topic composition in the case of news articles — 30% science and 70% politics, for example — they output each game level's gameplay type composition in the case of controller inputs. Moreover, we can discover all of this in an unsupervised manner. Developers can use this information to verify that their levels feature the appropriate style of gameplay and to recommend levels with gameplay that is similar to levels that players like.

The types of gameplay inherent in a game are not the only factors that determine the

controller inputs entered by the player, however. Other factors such as randomness within the game and the choices that players make can affect the controller inputs as well. In the latter case, a player might prefer fighting to sneaking when given the choice, so when they are playing a level that gives both choices they are more likely to enter input words that are fighting-related than sneaking-related. With this insight we developed the *player–gameplay action (PGA) model*, a novel extension of LDA that allows us to learn what gameplay types are present in a game in a way that is independent of each player's play style. We train a player recognition system on the PGA model's output to verify that its discoveries about gameplay are in fact independent of each player's play style. Our system can recognize a player from a database of eight players with over 90% accuracy in about 20 seconds of playtime, even for levels that the player has never played before and even when the controller is passed from another player.

Games as Moore Finite State Machines

Our choice to mine controller inputs and not some other signal to understand gameplay can be best explained by thinking of video games as Moore finite state machines (FSMs), which we depict in Figure 3.1. Here we see the feedback loop from which players' experiences are generated: the graphics and sound that are output by the game influence the player's next action. By observing different nodes and edges in the figure, we can learn about different aspects of the experiences that players have with games. Affect measures such as player surveys or skin conductance observe the Player node, for example, while vision and audio processing algorithms observe the Graphics and Sound nodes.

Figure 3.2: Nintendo Entertainment System (NES) controller. It features eight buttons: Up (U), Down (D), Left (L), Right (R), START (S), SELECT (E), B (B), and A (A). In *Super Mario Bros. 3*, the focus of our case study, Left and Right move Mario; the A button makes Mario jump, swim, float, or fly; and the B button makes Mario accelerate, throw fireballs, and open treasure chests. The START button pauses the game and the SELECT button moves the cursor in menu screens.

Today's game analytics systems observe the edge circled in purple, where games transition from state to state and generate event logs. But the point at which the player acts upon the game, and where the player's actions are most salient, is the edge circled in green: the player's controller inputs. By observing this edge, we can begin to understand and describe gameplay in a quantitative way. Of course, understanding gameplay fully would only be possible if we also measured what players are feeling in response to the game's output. We therefore view our focus on controller inputs as a first step towards making the evaluation of gameplay more objective. The methods that we present can complement those based on event logs and those based on the game's output to paint a fuller picture of the experiences that players have with games.

3.1 Defining the Concept of Input Words

Suppose we have a continuous stream of controller inputs as shown in Figure 3.3. Each row corresponds to a different button on the Nintendo Entertainment System (NES) controller (shown in Figure 3.2), and shaded bars mark the intervals when the buttons are pressed. Such a stream is a rich signal of what the player is doing, but we must somehow extract units of gameplay from this signal for it to be useful. How, then, can we represent these inputs in a way that encapsulates a meaningful notion of gameplay and is compatible with probabilistic inference methods?

This problem is nontrivial, and several properties of input streams make the problem even more challenging. First, input streams are noisy: players often press buttons on accident, and the exact timings of their button presses and releases do not matter. Whether a player makes Mario jump nine or ten frames after starting to accelerate, for example, makes no difference — they both represent the same semantic action. Second, input streams come in surges: although the NES receives inputs at 60 FPS, there are periods of very few inputs — or even no inputs — and periods of many inputs, even within a single level. Moreover, common actions such as running to the right or pressing no buttons at all reveal very little about a game's gameplay.

We solve this problem with the concept of *input words* that are analogous to words in natural language processing and visual words in computer vision. First, we define an *input state* to be the set of input buttons pressed during a given input frame. For example, "RBA" means that the player pressed Right, the B button, and the A button at the same time during an input frame. The input state "_" represents no buttons pressed. Then, we define

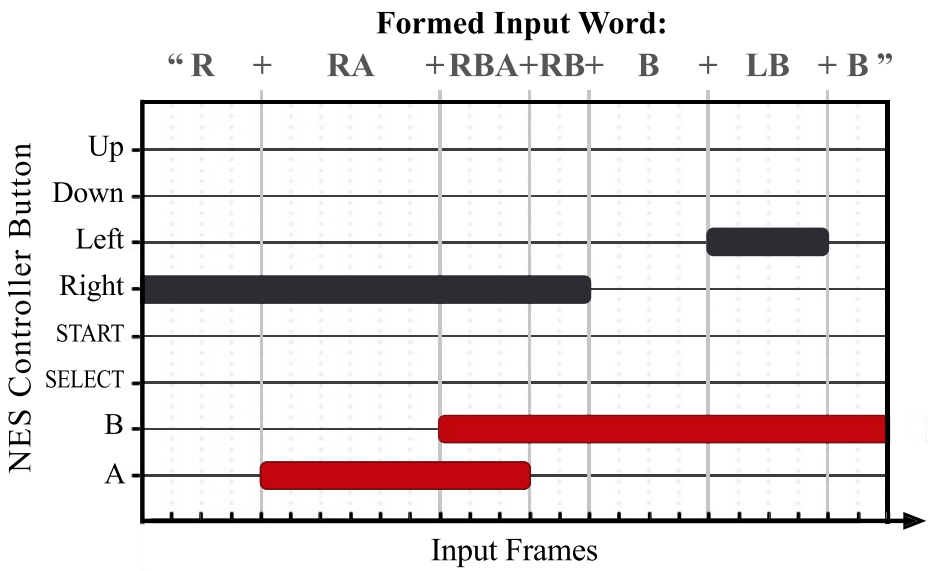


Figure 3.3: Forming input words. The shaded bars mark the intervals when buttons on the NES controller are pressed. The colors of the bars correspond to the colors of the buttons. There are seven input states over the course of this time interval: R, RA, RBA, RB, B, LB, and B. This sequence of inputs can therefore be represented with the input word “R+RA+RBA+RB+B+LB+B.” NES games run at 60 FPS.

an *input word* to be a sequence of input states entered over a fixed number of contiguous input frames called the word’s “extent” — in our case 30 frames, or half a second worth of controller inputs. The sequence of inputs in Figure 3.3, for example, would be represented as the word “R+RA+RBA+RB+B+LB+B.”

Defining input words in this way helps us reduce the noise from input streams because the input words ignore the number of frames that each input state is held. If the player pressed Right for one, two, or even three frames longer than depicted in Figure 3.3, for example, the formed input word would be the same. We labeled very common or rare words as “stop words” to exclude them from our consideration. The common stop words comprised 24.4% of words by frequency and the rare stop words comprised 10.2%. The word “__” alone, representing no buttons pressed during a 30-frame (half a second) time interval, comprised 18.2% of all observed words. Finally, we set the “word step size,” the

number of frames between the start of one sampled word and that of the next, to be 10 frames. Since each of the sampled words is 30 frames long, the words overlap with one another.

Semantics of Input Words

Semantically, input words represent what we call *gameplay* (or *action*) *primitives*: small combinations of input states or "verbs" [Crawford 2012] that define the possibility space of what the player can do at any given time. In many ways, these actions combine to form the greater dynamics [Hunicke, LeBlanc, and Zubek 2004] of how the game plays out, so they can be thought of as *microdynamics* as well. Moreover, we can treat them as discrete variables in probabilistic models to represent units of gameplay. Below is a sample of input words we observed in our experiments and what they represent in the context of players' playthroughs of *Super Mario Bros. 3* [Nintendo 1988], the subject of our case study:

"RB+RBA+B+LB": Making a fast forward jump, then holding Left to stop Mario's momentum;

"R+_+A+_+A+RA+R": Making nimble swimming movements to squeeze past a row of Cheep Cheeps (enemy fish);

"A+_+A+_+LA+L+_+A": Wagging Raccoon Mario's tail while falling to slow his descent, enjoying a moment of freedom before returning to the ground.

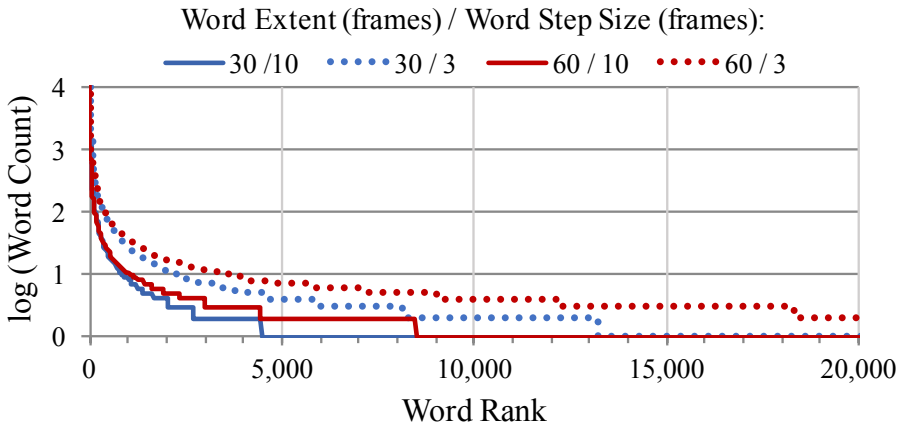


Figure 3.4: Word count distributions for representative input word extents and step sizes. The horizontal axis represents a count of one, the inverse logarithm of zero. The extent controls the number of frames that input words are sampled from, and the step size controls the number of frames between the start positions of consecutive words. The words were formed from about 15 hours of playtime. Since these curves have similar shapes, the vocabularies they represent have similar structures, and we can sample words coarsely without loss of generality. We chose a word extent of 30 frames and a step size of 10 frames — the solid blue line — for the experiments in the remainder of the paper. We label the top three ranking words and all words with counts less than ten — meaning log counts less than one — as stop words to exclude them from our analysis.

Choosing Word Extents, Step Sizes, and Stop Words

Earlier, we indicated that we formed each input word from 30 frames of input (meaning the "word extent" was 30 frames), that we began sampling a new word every 10 frames (meaning the "step size" was 10 frames), and that we labeled very common and rare words as "stop words." In order to set the parameters for word extent and step size, we searched the parameter space by forming input words from all of our recorded inputs using different configurations of these parameters. We then compared the distributions of word counts for each of the parameter configurations.

Figure 3.4 shows four representative distributions that correspond to different configurations of word extent and word step size. The words are ordered by decreasing count, and the horizontal axis represents a count of one — the inverse logarithm of zero. We note

that every distribution that we observed had the same general shape, suggesting that word extent and word step size do not affect the structures of input word vocabularies to a drastic degree. As a result, we elected to sample words more coarsely to make our algorithms run faster, setting the word extent and word step size to 30 and 10 frames, respectively. This corresponds to the solid blue line in Figure 3.4. Lastly, we labeled the top three words by count and all words with counts less then ten as stop words, leaving us with a vocabulary of 907 words.

3.2 Quantifying Gameplay with Topic Modeling

In the previous section, we saw that we can form input words from players' controller input streams to reveal *gameplay* (or *action*) *primitives* — what the player is doing from moment to moment in a video game. In this section, we show how to use input words to discover the types of gameplay that a game fosters and the extent that each game level fosters each type of gameplay, all in an unsupervised manner.

Using input words to discover gameplay types is challenging for two reasons: (1) the relationship between observed controller inputs and gameplay types is not obvious; and (2) it is difficult to objectively define which gameplay types are present in a game and to what extent they are present in different parts of the game. To address these challenges, we treat the problem of discovering gameplay types from controller inputs as a case of probabilistic topic modeling, which allows us to discover "topics" that represent gameplay types from players' recorded controller inputs. More specifically, we use latent Dirichlet allocation (LDA) [Blei, Ng, and Jordan 2003] — the simplest and most popular probabilistic topic

model — to model why some input words are entered by players and others are not. Our choice of LDA stems from four observations:

1. Input words recorded during a player's playthrough of a level can be combined into a "document" that represents what happened in that level;
2. Input words can show us what types of gameplay a level fosters;
3. Each level can be represented as a mixture of various gameplay types; and
4. Semantically similar input words tend to occur together.

This last observation forms our working definition of a gameplay type: a set of frequently co-occurring input words. Our experimental results in the remainder of this section will show that this definition is both intuitive and valid. Given this definition of "gameplay type," we can model each gameplay type in a quantitative way as a probability distribution over input words. Moreover, we will be able to quantify the extent that each gameplay type is present in each part of the game. Neither form of understanding is possible with existing game analytics methods. We did not use a deep learning approach to discover gameplay types because the interpretability of what we find — what the gameplay types mean — matters as much as the discovered gameplay types themselves.

Latent Dirichlet allocation (LDA)

Figure 3.5 shows the graphical model (Bayesian network) for LDA and Figure 3.6 shows the generative process for LDA — how LDA models which controller inputs players enter. Our description here will focus on what the variables in LDA mean in the context of gameplay and controller inputs.

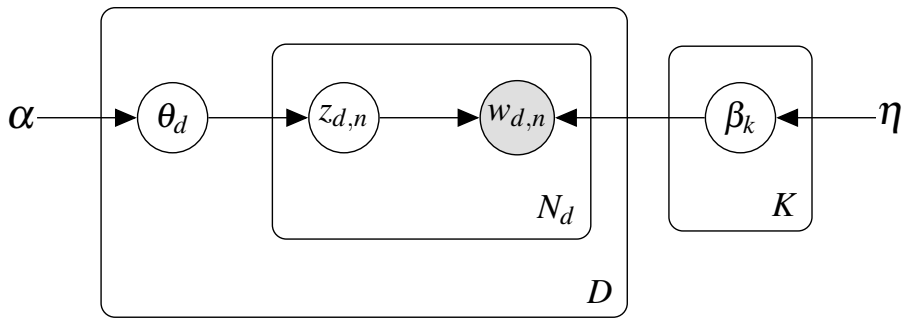


Figure 3.5: Graphical model for latent Dirichlet allocation (LDA). The generative process is shown in Figure 3.6. In our application of LDA to controller inputs, each document d is a gameplay session of a game section and θ_d are the gameplay type mixture proportions for that section. $z_{d,n}$ is the gameplay type assignment for word $w_{d,n}$, the nth word in document d. $\beta_{1:K}$ are the gameplay types ("topics").

The K gameplay types β_k are synonymous with topics and are represented as probability distributions of input words. Each document d is a gameplay session for a game section, and the game section exhibits multiple topics β_k with mixture proportions θ_d. The N_D input words from that document are categorically chosen from $\overline{\beta_d}$, the weighted average of the gameplay types that the game section exhibits:

$$\overline{\beta_d} = \sum_{k=1}^{K} \theta_d(k) \cdot \beta_k. \tag{3.1}$$

We call $\overline{\beta_d}$ the *effective gameplay type* of the game section represented by d. Here, $\theta_d(k)$ is the kth element of θ_d.

To compare LDA with the PGA model that we propose later, we define $\overline{\phi_{s,p}}$ to be the categorical distribution that the input words from player p's playthrough of game section s are effectively drawn from. For LDA, this is simply $\overline{\beta_d}$ for the document d representing s:

$$\overline{\phi_{s,p}^{\text{LDA}}} = \overline{\beta_d}. \tag{3.2}$$

$w_{d,n}$ is the nth word of document d and $z_{d,n} \in \{1{:}K\}$ is the random gameplay type that that word is drawn from. The presence of the z variables make representing $\overline{\beta_d}$ more tractable.

The boxes that surround the nodes in Figure 3.5 are called plates and represent replication — there are D different copies of the node θ_d, for example. The input words are all that we observe, so the $w_{d,n}$ node is the only one that is shaded. We perform posterior inference on the unshaded nodes to infer what their values are likely to be. The nodes that we care about most are $\theta_{1:D}$, the mixtures of gameplay types for each section of the game; and $\beta_{1:K}$, the input word distributions of the gameplay types themselves. We are able to infer the values of these nodes in an unsupervised manner. We set the hyperparameters α and η to 0.2 and 0.1, respectively. They control the respective sparsities of the documents' mixture proportions and the gameplay types' word distributions.

LDA Experimental Procedure

We analyzed the NES platforming game *Super Mario Bros. 3* (*SMB3*) as a case study, and we chose *SMB3* for three reasons: (1) unlike other genres such as simulation games and role-playing games (RPGs), the player's controller inputs directly control the main character in platforming games; (2) *SMB3* is one of the highest grossing video games of all time; and (3) the NES controller has just two action buttons, making the problem of recognizing patterns in inputs more challenging.

We invited a person who is experienced with *SMB3* to perform an entire playthrough of the game while we recorded her inputs to form a data set to provide as input to LDA. In all, we recorded all 91 of *SMB3*'s stages and five of its six types of special areas[1]. We

[1]The six types of special areas in *SMB3* are world map screens, Toad's houses, Spade minigames, N-

Generative Process for LDA

1. For each of K gameplay type ("topic") indices k:
   a) Draw $\beta_k \sim \text{Dirichlet}(\eta)$
2. For each of D documents d:
   a) Draw $\theta_d \sim \text{Dirichlet}(\alpha)$
   b) For each of the N_d words in document d:
      i. Draw $z_{d,n} \sim \text{Categorical}(\theta_d)$
      ii. Draw $w_{d,n} \sim \text{Categorical}(\beta_{z_{d,n}})$

Figure 3.6: Generative process for latent Dirichlet allocation (LDA). The graphical model is shown in Figure 3.5. Step 2b draws words $w_{d,n}$ from the document's effective gameplay type $\overline{\beta_d}$ as defined in Equation 3.1.

recorded the inputs over six play sessions lasting a total of roughly eight hours. In cases where it took the player multiple attempts to complete a stage, we concatenated the inputs from each attempt to form the final "document" for that stage.

We used the Bizhawk emulator [TASVideos community 2016] to record inputs for this experiment. We wanted the ability to play back previously recorded inputs so we could learn what different input words and gameplay types looked like in the game. Since the process of playing back players' controller inputs is sensitive to CPU load times, we had to use a frame-perfect emulator in order to play back these inputs properly. Bizhawk is considered by the tool-assisted speedrun (TAS) community to be the definitive frame-perfect NES emulator. The player recognition system that we develop later in this paper uses an input recording program that we built ourselves.

To perform posterior inference, we ran four separate chains of a No-U-Turn sampler (NUTS) [Homan and Gelman 2014] — a Markov chain Monte Carlo (MCMC) method — for 4,000 iterations apiece, with a burn-in period of 2,000 iterations and a lag of one

Mark Spade minigames, Hammer Bros. stages, and the legendary Treasure Ship, this last of which was not encountered during our player's playthrough.

iteration. The latent variables that we care about most are $\theta_{1:D}$, the mixtures of gameplay types for each section of the game; and $\beta_{1:K}$, the input word distributions of the gameplay types themselves. We are able to infer the values of these nodes in an unsupervised manner. We set α to 0.2, η to 0.1, and K (the number of gameplay types) to 12 after inspecting the results from a few shorter trial runs. The sampler took over two days to run on a modern machine.

LDA Experimental Results

The two primary outputs from LDA are the game levels' gameplay type mixture proportions $\theta_{1:D}$ (Figure 3.7) and the discovered gameplay types $\beta_{1:K}$ (Table 3.1), both of which are unshaded in Figure 3.5. Put differently, $\theta_{1:D}$ (Figure 3.7) depicts the extent that each of *SMB3*'s stages fosters each of the discovered gameplay types ("topics"), while $\beta_{1:K}$ (Table 3.1) depicts the discovered gameplay types themselves. Figure 3.7 is displayed in heat map form where the rows represent stages and the columns represent the twelve gameplay types. The fact that there are many dark blues in the figure means that the mixtures are relatively sparse and, as a result, that each stage exhibits a small number of gameplay types — a desirable property for a topic model to have. Table 3.1, to be precise, shows a sample of $\beta_{1:K}$: the top 15 input words in four of the discovered gameplay types.

In a traditional context, we could understand what each topic represents just by examining the words in their column in Table 3.1. A topic featuring the words "party," "election," "debate," and "primary," for example, could be understood to be about politics. Here, however, it is not so easy to understand what the gameplay types represent because the

relationships between each topic's input words are not so obvious. What does Gameplay Type 5 here mean, for example? To answer this question, we find examples of stages that strongly exhibit the gameplay type using Figure 3.7, then we watch gameplay videos of those stages — which we recorded using Bizhawk earlier — to see how that gameplay type's input words are manifested in those stages. In the case of Type 5, we find that that topic is prominent in nine of *SMB3*'s stages[2], four of which are shown in Figure 3.9. Although these stages look very different from each other at first glance, their gameplay videos reveal that they all require players to jump around flying enemies or projectiles such as Flying Cheep Cheeps or Podoboos. Type 5, therefore, seems to represent this type of gameplay.

Likewise, Type 12 is exhibited by stages that require repeatedly pressing the A or B button respectively while stationary. Figure 3.8 shows two such cases: World 6-7, in which Fire Mario and throws lots of fireballs; and World 1-6, in which Raccoon Mario lands softly with his tail.

Together, Figure 3.7 and Table 3.1 allow us to describe the types of gameplay in *SMB3* in a quantitative way. As a point of comparison, a topic model applied to a news corpus might show that a news article has a blend of 30% science and 70% politics. Likewise in our case, Figure 3.7 shows that World 3-2 has a blend of 35% Gameplay Type 5 (jumping around flying enemies and projectiles) and 23% Gameplay Type 3 (making precise running jumps onto narrow platforms). Since Worlds 3-3 and 6-3 have very similar blends of gameplay types (Figure 3.7), players who like the type of action in World 3-2 should enjoy these

[2]The nine stages that strongly exhibit Type 5 are Worlds 3-2, 3-3, 3-7, 3-8, 5-9, 5-Fortress2, 6-3, 6-5, and the Hammer Bros. battles.

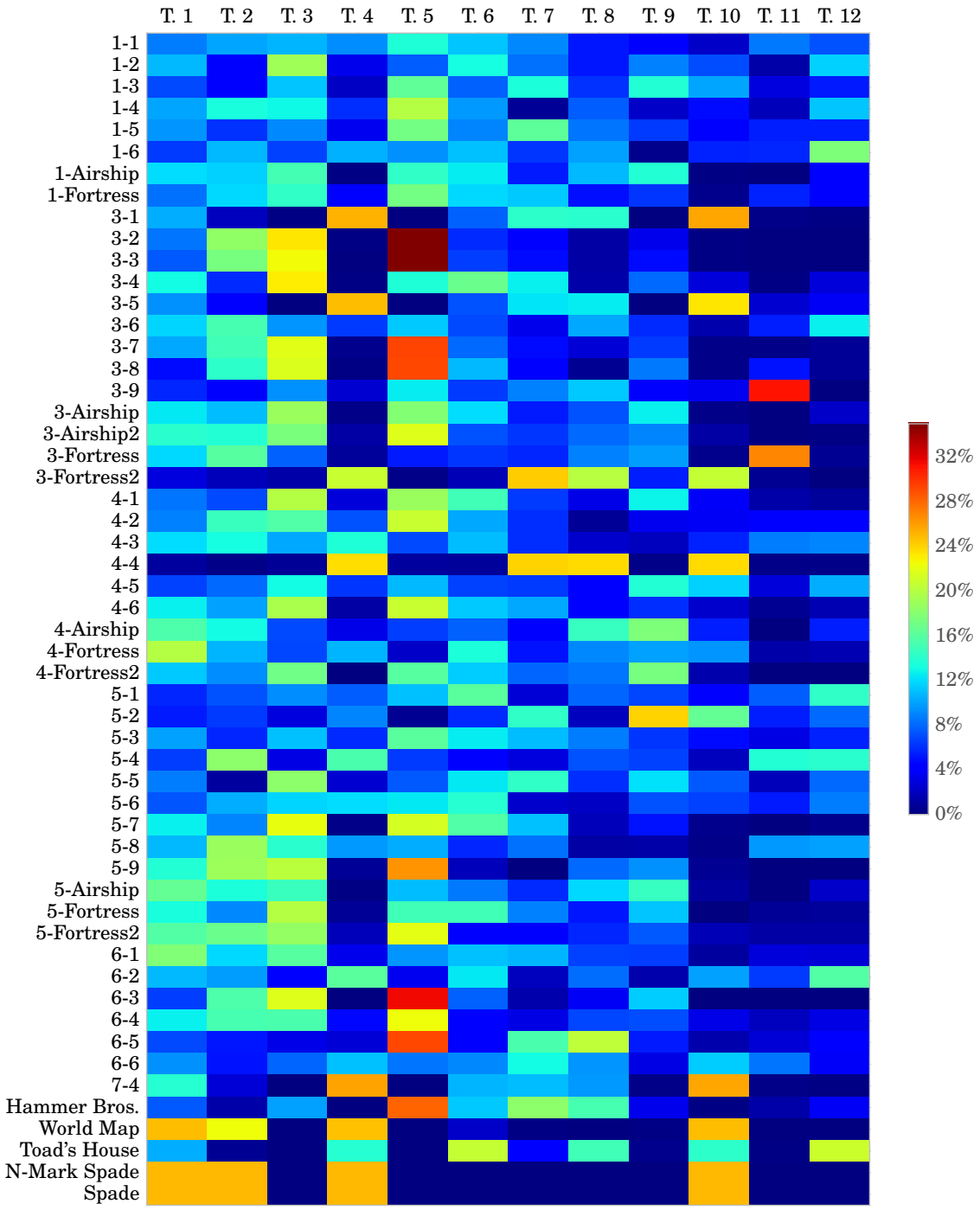
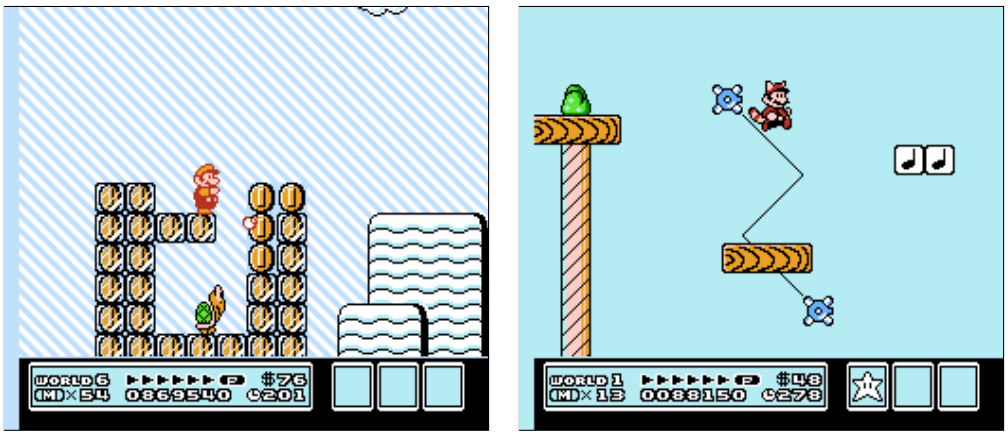


Figure 3.7: Inferred gameplay type mixture proportions for selected stages in *Super Mario Bros. 3* as output by LDA. Each row represents a stage and sums to 100%. The columns represent gameplay types. The bottommost rows represent the five types of special areas, which are easy to differentiate from the regular stages. Stages with high proportions of Types 4 and 10 are underwater stages. A recommendation system can use this information to recommend stages with similar gameplay to ones that players like.



(a) World 6-7 (b) World 1-6

Figure 3.8: Stages representative of Gameplay Type 12 from Figure 3.7 and Table 3.1. In (a) Mario throws fireballs to thaw coins frozen in ice, and in (b) Raccoon Mario lands softly by wagging his tail. Both stages encourage pressing an action button repeatedly while stationary.

stages as well. This type of understanding is new to our method and cannot be found from event logs. Using this approach, developers can learn if their game has enough variety to keep players' interest, see which types of gameplay are most popular with players, check to see if a level fosters a desired gameplay type, and build recommendation systems to help players find levels that play similarly to ones they like.

Figure 3.7 can also help us discover *outlier stages*: stages in *SMB3* that play very differently from the rest of the game. The world map, Spade minigames, and N-Mark Spade minigames at the bottom of Figure 3.7 are three such stages since their gameplay type mixtures that are unlike the rest. These stages consist of players choosing items from menus. Likewise, *SMB3*'s underwater stages[3] can also be considered outliers. These stages strongly exhibit Gameplay Types 4 and 10, which are included in Table 3.1. Many players dislike playing underwater levels because of how different they feel to play [WeFightFor-

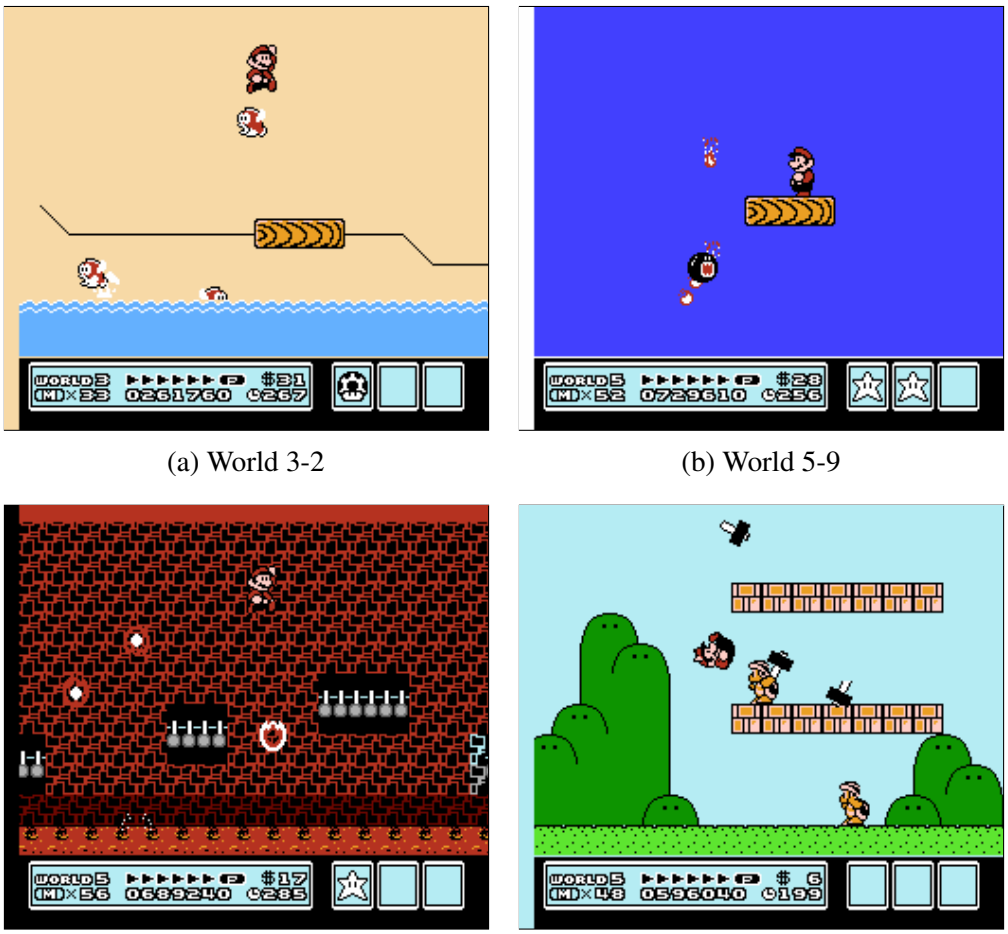
[3] *SMB3*'s underwater stages are Worlds 3-1, 3-5, 3-Fortress2, 4-4, and 7-4. World 6-6 also has a substantial underwater component.

Type 4	Type 5	Type 10	Type 12
R	LB	R	B+_
_+A+_	LB+B	_+A+_	DB
_+R	B+RB+RBA	_+R	_+B
R+RA+R	B+LB	D	RA+R+RA+R+RA+R+RA
_+R+_	LB+LBA	R+RA+R	_+B+_
A+_	LB+B+RB	_+R+_	_+A
RA+R+RA+R+RA+R+RA	B+LB+B	A+_	_+R+RA
R+_	LBA+LB	R+_	R+RA+R+RA+R+RA+R
_+A	B+RB	_+L+_	_+A+BA+B+_
_+L+_	RBA+RB+B+LB	_+A	RA+R+RA+R+RA+R
_+L	RBA+RB	DB	_+B+_+B+_+B+_
R+RA	RBA+BA+B+LB	_+L	B+BA
R+RA+R+RA+R+RA+R	RB+B+LB	D+_	R+RA+R+RA+R+RA
RA+R+RA+R+RA+R	LB+B+RB+RBA	L+_	RBA+RB+B
R+RA+R+RA+R+RA	B+RB+B	_+D+_	B+_+B

Table 3.1: Inferred gameplay types ("topics") from LDA for one player's complete playthrough of *Super Mario Bros. 3*. We show the top 15 input words in each gameplay type in order of decreasing frequency. Type 5 relates to jumping around flying enemies or projectiles, as Figure 3.9 shows. Type 12 seems to relate to pressing action buttons repeatedly to land softly as Raccoon Mario or melt ice with fireballs, respectively. Type 10 relates to basic directional movements such as walking slowly and making menu selections.

ever 2015], and the results in Figure 3.7 support that assertion. Interestingly, only half of World 6-6 is underwater, and that stage exhibits lesser amounts of Types 4 and 10. By examining our recorded gameplay videos, we see that Type 4 captures short, nimble hops in land levels and nimble swimming movements in water levels while Type 10 captures slower, more gradual movements on land and in water.

Finally, Figure 3.7 can help us infer the specific choices that our player made within the game, as long as those choices manifest themselves in the input words that we observe. For example, we can infer whether a player completes an underwater level the hard way or wears a Frog Suit instead by examining the mixture proportions of Gameplay Types 7 and 8. Figure 3.10 shows gameplay from both cases. In the latter case, the proportions for Gameplay Types 7 and 8 increase to twice their regular values. Of course, we can discover the answer to this particular question by simply recording event logs, and we argue that



(a) World 3-2

(b) World 5-9

(c) World 5-Fortress2

(d) Hammer Bros. stage

Figure 3.9: Stages representative of Gameplay Type 5 from Figure 3.7 and Table 3.1. Although these stages look quite different at first glance, they all share the same primary type of gameplay: jumping around flying enemies or projectiles. Here we see (a) Flying Cheep-Cheeps, (b) Fire Chomps and their fireballs, (c) Podoboos, and (d) Hammer Bros. *Super Mario Bros. 3* © Nintendo.

controller inputs should complement — and not replace — such information. We include this example, however, to show how expressive controller inputs can be.

3.3 The Player–Gameplay Action (PGA) Model

In the previous section, we showed how to use LDA to understand the types of action present in a video game from streams of recorded controller inputs. Although LDA yielded

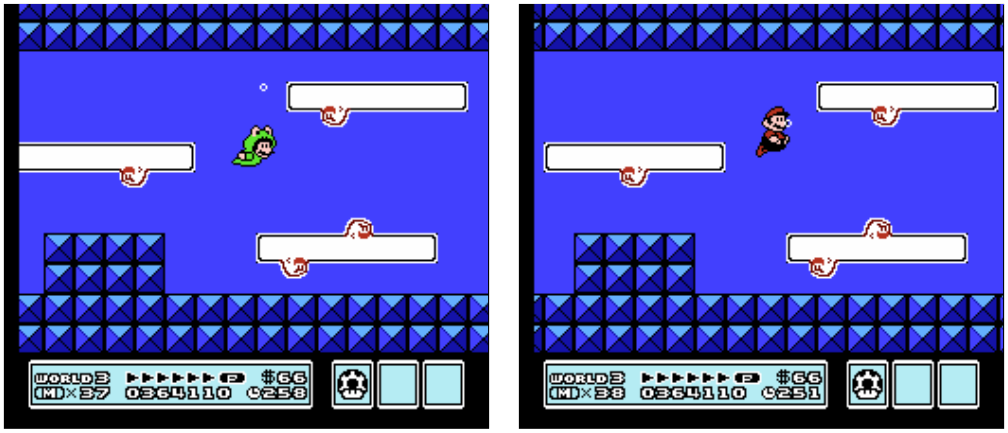


Figure 3.10: World 3-Fortress2 with and without the Frog Suit, respectively. The Frog Suit allows Mario to swim in the water much more easily. This stage is difficult and culminates in a mini-boss fight, so many players save a Frog Suit to wear for the occasion. By examining the gameplay type mixtures of a playthrough of this level, we can tell whether the player wore the Frog Suit or completed it the hard way. *Super Mario Bros. 3* © Nintendo.

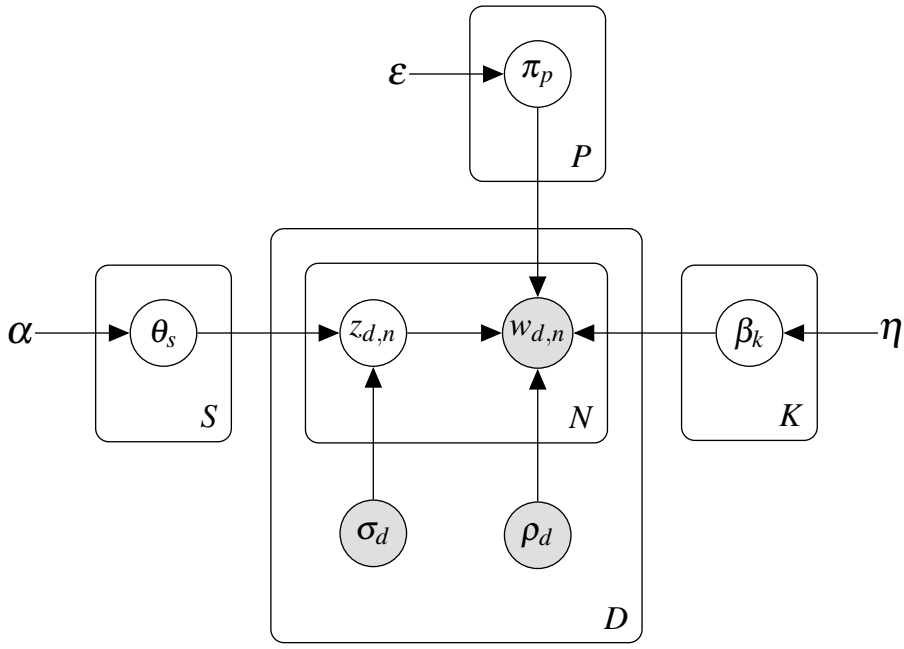


Figure 3.11: Graphical model for the player–gameplay action (PGA) model. The generative process is shown in Figure 3.12. Each document d is a gameplay session for a particular player p as indicated by ρ_d and a particular section of the game s as indicated by σ_d. θ_s are the gameplay type proportions for section s of the game. $z_{d,n}$ is the gameplay type assignment for word $w_{d,n}$, the nth word in document d. $\beta_{1:K}$ are the gameplay types and $\pi_{1:P}$ are the players' play styles.

meaningful results, one of its shortcomings is that it assumes that the stages' gameplay type mixture proportions and the gameplay types themselves are the *only* factors that affect which controller inputs players enter. In reality, other factors such as (a) randomness within the game, (b) the unique way that each player holds the controller and presses buttons, and (c) the choices that players make can affect the controller inputs as well. Regarding the last case, suppose a game level gives players the choice between two types of gameplay: sneaking past enemies or fighting them head-on. Such a level can be modeled as a mix of 50% sneaking and 50% fighting, assuming players are equally likely to make either choice. If we observed the controller inputs from a single player's playthrough of the level, however, we might infer that the level comprises 100% of the gameplay type that the player chose to play instead of the more accurate representation of 50% sneaking and 50% fighting.

With this insight we propose the *player–gameplay action (PGA) model*, a novel extension of LDA that allows us to understand the types of action present in a video game in a way that is independent of each player's play style. This corrects for both (b) and (c) in the previous paragraph. Figure 3.11 shows the graphical model for the PGA model and Figure 3.12 shows the generative process for the PGA model. S is the number of sections in the game, P is the number of players, and D is the total number of recorded gameplay sessions. We use the generic term "section" instead of "stage" or "level" here because the model is also applicable to games that are not divided into traditional stages or levels. In LDA, each document d is a gameplay session for a specific section of a game, but in the PGA model each document d is a gameplay session for a specific section of a game *played by a specific player*. $\sigma_d \in \{1{:}S\}$ and $\rho_d \in \{1{:}P\}$ indicate the gameplay section being played

Generative Process for PGA

1. For each of K gameplay type ("topic") indices k:
   a) Draw $\beta_k \sim \text{Dirichlet}(\eta)$
2. For each of the S game sections s:
   a) Draw $\theta_s \sim \text{Dirichlet}(\alpha)$
3. For each of the P players p:
   a) Draw $\pi_p \sim \text{Dirichlet}(\varepsilon)$
4. For each of the D gameplay session documents d:
   a) For each of the N words in document d:
      i. Draw $z_{d,n} \sim \text{Categorical}(\theta_{\sigma_d})$
      ii. Draw $w_{d,n} \sim \text{Categorical}\left(\frac{\beta_{z_{d,n}}\pi_{\rho_d}}{|\beta_{z_{d,n}}\pi_{\rho_d}|}\right)$, where $|\beta_{z_{d,n}}\pi_{\rho_d}|$ is a normalizing constant

Figure 3.12: Player–gameplay action (PGA) model generative process.

and the player who was playing during gameplay session d, respectively. β, θ, α, and η are unchanged from LDA. We set ε to be 0.1 like α.

Two factors control the input words that are generated in the PGA model: the gameplay types $\beta_{1:K}$ and the players' play styles $\pi_{1:P}$, which are both represented as categorical distributions over input words. Figure 3.13 shows how we model their interaction. The main idea is that the two distributions act as filters on each other so that the input words are generated from the (normalized) product of the two distributions. A player's fighting-oriented play style, for example, would filter out input words related to sneaking from a game section that exhibits both fighting and sneaking. Likewise, a fighting-oriented game section would filter out input words related to sneaking from the play style of a player who is indifferent to fighting and sneaking. In equation form, the PGA model redefines $\overline{\phi_{s,p}}$ from Equation 3.2 to be:

$$\overline{\phi_{s,p}^{\text{PGA}}} = |\overline{\beta_s}\pi_p|. \tag{3.3}$$

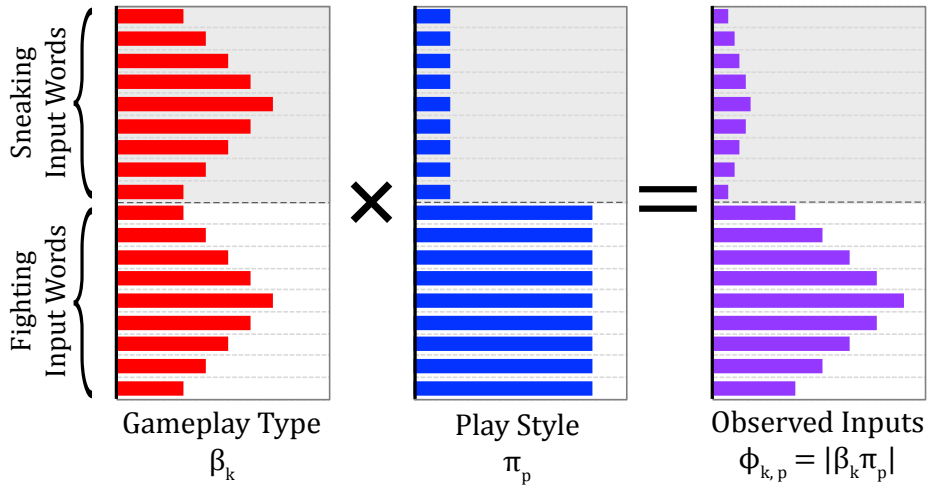


Figure 3.13: Player–gameplay interaction in the PGA model. The distributions above are distributions over input words. LDA assumes that input words are generated from a game's gameplay types $\beta_{1:K}$ — the red distribution only — ignoring the players' play styles $\pi_{1:P}$. The PGA model, however, assumes that gameplay types and players' play styles act as filters on each other, and that input words are generated from the (normalized) product of the two. In the example shown here, a player's fighting-oriented play style dampens sneaking-oriented input words from a gameplay type that exhibits both fighting and sneaking.

$\overline{\beta_s}$ is defined in Equation 3.1, in which d is used in place of s.

PGA Model Experimental Procedure

To analyze the PGA model's performance, we recorded controller inputs from eight players playing *SMB3*. For Player 1, we used the same data as we did in our LDA experiment: a full playthrough of *SMB3*, 91 regular levels plus 5 types of special areas, totaling 96 stages. For the other players, we used data from their playthroughs of World 1, which total eight stages and five types of special areas each. Players 1, 2, 3, and 5 were very experienced with platforming games, Players 4 and 8 were somewhat experienced, and Players 6 and 7 were not at all experienced.

We estimated the latent variables $\theta_{1:S}$, $\beta_{1:K}$, and $\pi_{1:P}$ by running four NUTS chains for 2,000 iterations apiece, with a burn-in period of 1,000 iterations and a lag of one iteration.

Type 3	Type 7	Type 9	Type 11
_+A+_	R	_+R+RA	L
LB	RA+R	LB	_+R+RA
_+R+_	_+A+_	RB+RBA	RA
A+_	R+RA+R	DB	URA
_+L+_	R+RA	RA+R+_	R+RA
LB+B	A+_	LB+B+RB	RA+R+_
LB+LBA	UR	B+DB	RA+R+RA+R+RA+R+RA
B+LB	_+R+_	RA+R+RA+R+RA+R+RA	RB+RBA
_+A	R+UR	L+_+R	_+R+UR
_+D+_	_+A	URA	UR+URA
_+R+_+R	R+_	RA	L+LA+L
LBA	_+L+_	RBA+RB	RA+R+_+L
LBA+LB	R+_+L	RA+R+_+L	_+R+UR+URA
RB+B+LB	_+D+_	DB+B	_+UR+URA
R+_+R+_	RA+R+_	UR+URA	R+RA+R+RA+R+RA+R

Table 3.2: Inferred gameplay types ("topics") from the PGA model. This table is red to match β_k in Figure 3.13. We show the top 15 input words in each gameplay type in order of decreasing frequency. These gameplay types correspond to the ones in Figure 3.14.

We set K to 12 as we did in our LDA experiment. The sampler took several days to run on a modern machine.

PGA Model Results

The PGA model infers three things at once, providing them as output: the game levels' gameplay type mixture proportions $\theta_{1:S}$ (Figure 3.14), the discovered gameplay types $\beta_{1:K}$ (Table 3.2), and the discovered play styles $\pi_{1:P}$ of players (Table 3.3). Recall that the first two were output by LDA and that the third is modeled and output here to remove its effect from the first two. Our focus here will be on seeing if the PGA model outputs more accurate depictions of $\theta_{1:S}$ (Figure 3.14) and $\beta_{1:K}$ (Table 3.2) than LDA did in Figure 3.7 and Table 3.1.

Starting with $\theta_{1:S}$, we see that the PGA model's output mixture proportions in Figure 3.14 have a greater range than LDA's in Figure 3.7, but the sparsity of each stage's

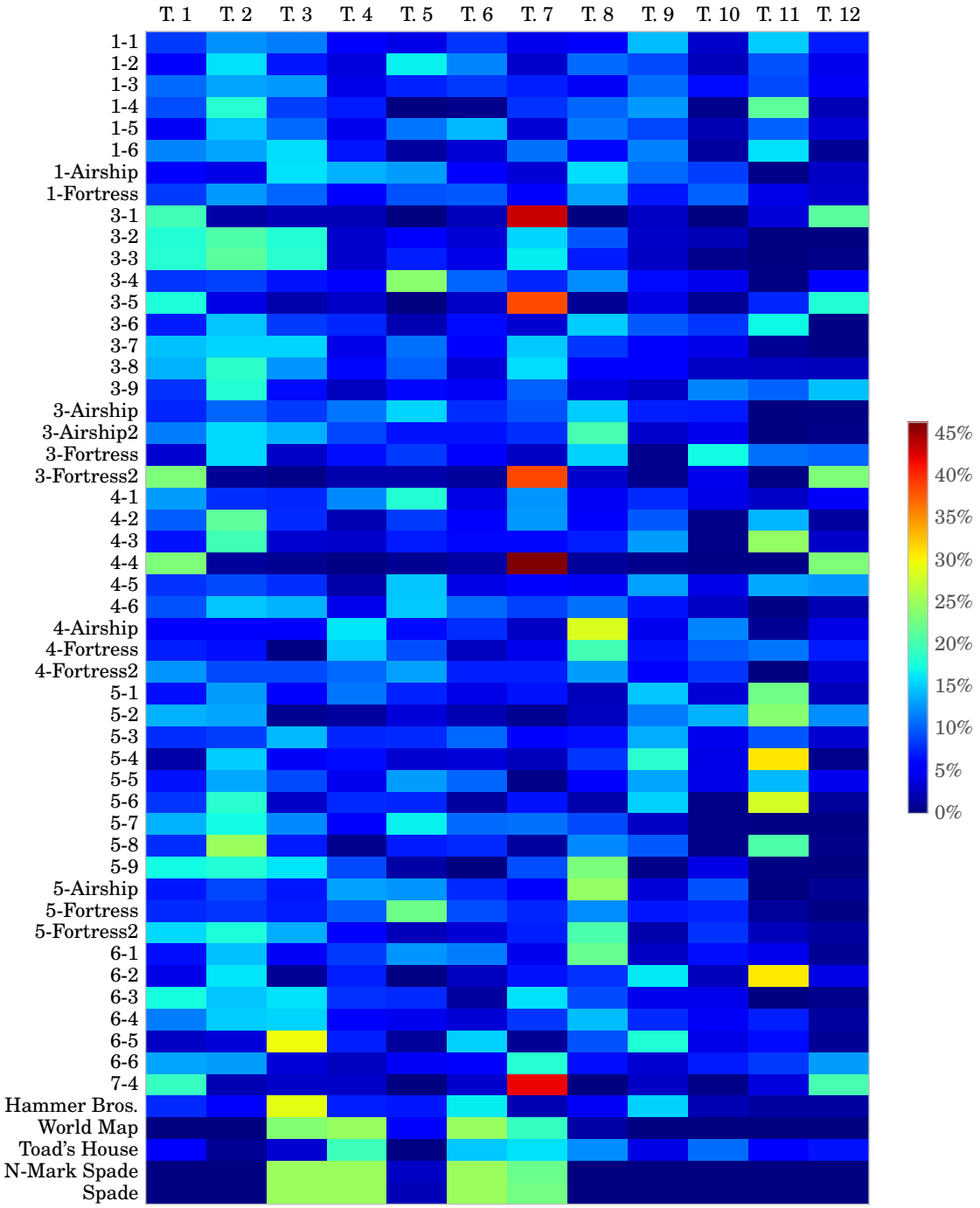


Figure 3.14: Inferred gameplay type mixture proportions for selected stages as output by the PGA model. The rows represent stages and the columns represent gameplay types. Each row sums to 100%. The values here have a greater range than those output by LDA in Figure 3.7 and the topics seem easier to interpret. Gameplay Types 7 & 12 seem to indicate underwater stages, Type 11 represents flying or floating, Type 9 represents making long running jumps, and Type 3 represents backtracking.

mixture proportions as measured by their range and standard deviation is not significantly different between the two. Moving to $\beta_{1:K}$, we find that the coherence scores [Mimno, Wallach, Talley, Leenders, and McCallum 2011] for the top 30 words in the PGA model's output gameplay types (M: -299.9, SD: 40.7) are significantly worse on average than those for LDA's gameplay types (M: -189.1, SD: 46.4); $t_{11} = 6.34$ ($p < 0.0001$). A coherence score is a popular measure of topic quality and measures how often a topic's most frequent words occur in documents together.

In our experience, however, we found that the PGA model's gameplay types in Table 3.2 were easier to interpret than LDA's in Table 3.1. Types 7 and 12 in the PGA model, for example, seem to indicate underwater stages even more strongly than Types 4 and 10 from LDA did. Within minutes of examining gameplay videos, we were able to interpret the general meaning of most of these gameplay types. Type 2 is akin to Type 5 from LDA, Type 11 represents flying or floating with Raccoon or Tanooki Mario's tail, Type 9 represents making long running jumps, and Type 8 represents jumping on enemies, bosses, and Starman items that move around sporadically. Type 3 seems tuned to backtracking and is most salient in maze-like levels with lots of backtracking. We feel that coherence scores may not be suitable for the PGA model because, as Figure 3.13 shows, this model does not model input words as being generated directly from topics like LDA does.

Finally, Table 3.3 shows a sample of the players' play styles $\pi_{1:P}$, an output not present in LDA. Here we can see the distinguishing factors of each player's play style. Player 1 floats a lot compared to other players, for example, exhibiting input words of the form "R+RA+R+RA...". Moreover, the less experienced players' (Players 6–8's) play styles are much more basic than the other players', with much shorter input words. Together, Fig-

ure 3.14 and Table 3.3 suggest that the play style distributions $\pi_{1:P}$ capture the variations in controller inputs caused by each players' unique playing style, leaving the inferred gameplay types $\beta_{1:K}$ less affected by those variations. In other words, the PGA model allows us to understand the gameplay types present in *SMB3* in a way that is independent of each player's play style. The accuracy of our player recognition system in the next section will further this conclusion.

3.4 Player Recognition Using the PGA Model

The PGA model allows developers to categorize levels based on how they feel to play, build recommendation systems that account for players' tastes, adapt game content to players' individual play styles, and more. In this section, we demonstrate one such application: a player recognition system based on controller inputs alone. Although recognizing users from their inputs is by no means novel [Buckley, Chen, and Knowles 2013; Hurst, Hudson, and Mankoff 2007], we do this here to show that the PGA model can indeed control for the effect that a players play style has on the controller inputs he or she enters, allowing developers to understand the types of action present in a video game in a way that is independent of each players play style.

Our system represents each player p with his or her play style π_p. It is a simple classifier that computes the most likely player p given the input words w observed so far and the PGA model's outputs θ_s and $\pi_{1:P}$, where s is the game section being played. This likelihood can

Player 1	Player 2	Player 6	Player 7
LB	DB	R	R
RB+RBA	DB+B	RA+R	R+_+L
RA+R+RA+R+RA+R+RA	B+RB+B	R+_	L+_+R
LB+LBA	DB+B+DB	RA+R+_	L
B+RB	B+RB	_+A+_	L+LA
_+U+_	B+DB	_+L	_+L+_+R
B+LB	B+RB+B+RB	_+L+_	L+LA+L
RB+B	RB+B	L+_	R+RA
LBA+LB	DB+B+DB+B+DB+B+DB	R+RA+R	R+RA+R
LB+B+RB	DB+DRB+RB	R+RA	_+L
_+B	B+LB+B	_+R+_	R+_
R+RA+R+RA+R+RA+R	DB+B+RB	A+_	RA+R
_+R+RB	BA	_+R+RA	_+L+_
LB+B	RB+DRB+DB	_+A	R+_+R
RA+R+RA+R+RA+R	RB+B+RB	R+_+R	L+_

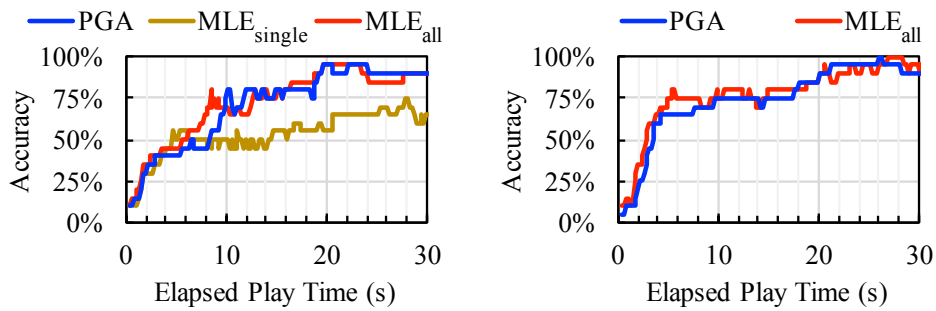
Table 3.3: Inferred play styles from the PGA model. This table is blue to match π_p in Figure 3.13. We show the top 15 input words in each play style π_p for four of our eight players, who are nicknamed Alice, Bob, Dave, and Eve respectively in Figure 3.16. These words are the ones that best distinguish each player, *not* just the ones that each player input the most. Players 6–8 were inexperienced with platforming games and have much more basic play styles than the other players do.

be computed as follows:

$$p(\rho = p \mid \theta_s, \beta_{1:K}, \pi_{1:P}, w) = \prod_{w_i \in w} \overline{\phi_{s,p}^{\text{PGA}}}(w_i), \tag{3.4}$$

where $\overline{\phi_{s,p}^{\text{PGA}}}$ is defined in Equation 3.3 and $\overline{\phi_{s,p}^{\text{PGA}}}(w_i)$ is the probability of w_i in the distribution $\overline{\phi_{s,p}^{\text{PGA}}}$. To test this classifier, we recorded a new set of inputs in which five of our eight players played eight stages each: four stages that they previously played during our PGA model experimental procedure and four stages that they did not play before. We compare this classifier with two others that use different approaches for calculating the categorical distribution parameters $\phi_{s,p}$. Classifier $\text{MLE}_{\text{single}}$ sets them to be "smoothed" versions[4] of the maximum likelihood estimates (MLEs) given the inputs player p previously made

[4]In reality, we use a Dirichlet–Categorical conjugate model with hyperparameter 0.1 to "smooth" out the MLE, ensuring that all of its elements are nonzero.



(a) Stages Played Before (b) Stages Never Before Played

Figure 3.15: Player recognition system accuracy over play time. We show this accuracy for (a) stages that players played before and (b) stages that players did not play before. We compare three recognition strategies. 'PGA' recognizes players using Equation 3.4 and the PGA model's outputs. '$\text{MLE}_{\text{single}}$' computes players' maximum likelihood estimates given their inputs from their previous playthrough of the stage being played — it does not apply for stages that players did not play before. 'MLE_{all}' computes the same but for all stages that players previously played. The PGA model-based system can recognize players with 95% accuracy after about 20 seconds of playtime — the same as if we used the 'MLE_{all}' approach — for both previously played stages and never before played stages.

on stage s, and Classifier MLE_{all} sets them to be the "smoothed" MLEs given the inputs player p made on all previously played stages. We simulated the same inputs across all three classifiers via remulation [Bi, Azenkot, Partridge, and Zhai 2013a].

Player Recognition Results

Figure 3.15 shows our three recognition systems' accuracy over time during a player's play session. Both the PGA model-based system ('PGA') and Classifier MLE_{all} ('MLE_{all}') are able to recognize the player with over 90% accuracy in roughly 20 seconds of playing, even for stages that he or she has never played before. The mean (std. dev.) accuracy between the 20 s and 30 s marks for stages the player has played before is 91.3% (12.9%) for 'PGA,' 65.3% (31.6%) for '$\text{MLE}_{\text{single}}$,' and 90.0% (15.5%) for 'MLE_{all}.' There is a significant main effect for the classifier used; $F_{2,8} = 7.29$ ($p = 0.016$). Pairwise mean

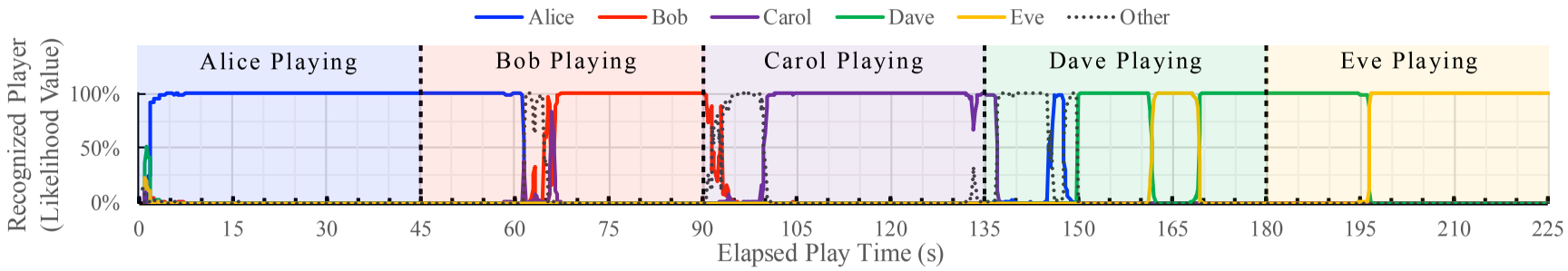


Figure 3.16: Player recognition system performance while players take turns playing. Our system is trained on the outputs from the PGA model and recognizes players from a sliding window of the last 25 seconds of controller inputs received. It was trained with eight players and can recognize each of these five in turn within 10–20 seconds of him or her taking the controls. We simulate the turn-taking by concatenating 45 seconds apiece of their controller inputs from their playthroughs of World 1-3 of *Super Mario Bros. 3*. These are not the inputs that we used when training the PGA model.

comparison showed that the differences were significant for every pair except 'PGA' vs. 'MLE_{all}.' For stages that the player did not play before, there is no significant difference between this accuracy metric for 'PGA' (M: 93.6%; SD: 8.5%) and 'MLE_{all}' (M: 93.1%; SD: 8.3%); $t_4 = 0.15$ ($p = 0.89$). These results show that the PGA model's outputs can power player recognition systems that work just as well as ones based on players' complete word frequency distribution histories. This means that the play styles output by the PGA model do in fact capture the unique playing styles of each player. Hence, the PGA model can indeed control for the effect that a player's play style has on the controller inputs he or she enters — unlike LDA.

Figure 3.16 shows how the 'PGA' recognition system performs when players take turns playing by passing the controller to each other. It can recognize each player in turn from a database of eight after he or she plays for 10–20 seconds. To simulate the turn-taking, we concatenated 45 seconds apiece of the five players' new playthroughs of World 1-3. This version of our recognition system computes its output from a sliding window of the last 25 seconds of inputs heard. The two players most confused for each other are Dave and Eve, who correspond to Players 6 and 7 in Table 3.3 and had no previous experience with platforming games. This suggests that novice players are harder to tell apart than expert players.

3.5 Discussion

In this chapter, we show that analyzing players' controller inputs using probabilistic topic models allows game developers to describe the types of gameplay — or action — in games

in a quantitative way. To do this, we define the concept of *input words* to represent gameplay primitives, then show how probabilistic topic models such as LDA can extract meaningful gameplay types from these primitives. To make the same types of discoveries about gameplay in a way that is independent of each player's play style, we develop the *player–gameplay action (PGA) model*, a novel extension of LDA. We train a player recognition system on the PGA model's output to verify that its discoveries about gameplay are in fact independent of each player's play style. It recognizes players with over 90% accuracy in about 20 seconds of playtime.

We describe the limitations of our methods and ideas for future work in Section 7.2.

Chapter 4

Gaze Locking: Passive Eye Contact Detection for Human–Object Interaction

This chapter explores how we can make computers sense attention in a lightweight and accurate way. By doing so we can make providing input (or commands) to computers much faster, an important part of this dissertation's goal of making using computers feel more like using powers of our own.

We will start with some theory. Each command that we give to a computer or device specifies three things, either explicitly or implicitly: the computer or device that must perform the action, the action that must be performed, and, if necessary, the object that the action should be performed on. These parts correspond to the components of an imperative sentence: the subject, verb, and direct object, respectively. In imperative sentences, the subject (meaning, the recipient of the command) is often the easiest to specify and, in fact, is usually a given. Take, for example, the following two commands: "Open the door" and "Turn off the light." In both cases, the subject is the implicit "you."

When it comes to commanding devices, however, specifying the subject is often the most time-consuming and laborious part. To turn on a TV, for example, a user must first find the particular remote that controls the TV, hold the remote, orient it the proper way in their hand, and point it toward the TV — all of this equates to specifying the TV as the subject of the command. To turn on a light, the user must first navigate to a particular light

switch — either a physical switch or, for smart lights, a particular screen on a particular app on the user's phone that controls the light.

Gaze as an Input Mechanism

A natural idea for making specifying the subjects commands much faster is to treat people's eye gaze as an input mechanism, allowing them to select the subjects of commands and perhaps the direct objects of commands as well just by looking at them. In theory, gaze input offers several advantages over other types of input: it is nearly instantaneous, can be done over long distances, and often requires no additional effort from the user since they would be likely look at what they intend to interact with anyway.

Gaze input also echoes how people communicate with each other: eye contact is a major form of nonverbal communication and people use it to seek information (e.g., to see how something we say is received), regulate interaction (e.g., by signaling when it is someone else's turn to speak), and much more [Argyle and Dean 1965; Kendon 1967; Vertegaal, Slagter, Veer, and Nijholt 2001]. Interestingly, while we have a hard time determining the exact angle at which someone else is looking, we seem to be very good at determining when someone else is looking *at us* (i.e., at or very near our eyes), and are acutely aware of it [Gemmell, Toyama, Zitnick, Kang, and Seitz 2000].

Most gaze-based interactive systems today rely on gaze tracking. They find the exact angle users are looking at instead of sensing eye contact directly. Although gaze tracking has been extensively studied and current methods [Baluja and Pomerleau 1994; Beymer and Flickner 2003; Hansen and Pece 2005; Morimoto, Amir, and Flickner 2002; Tan,

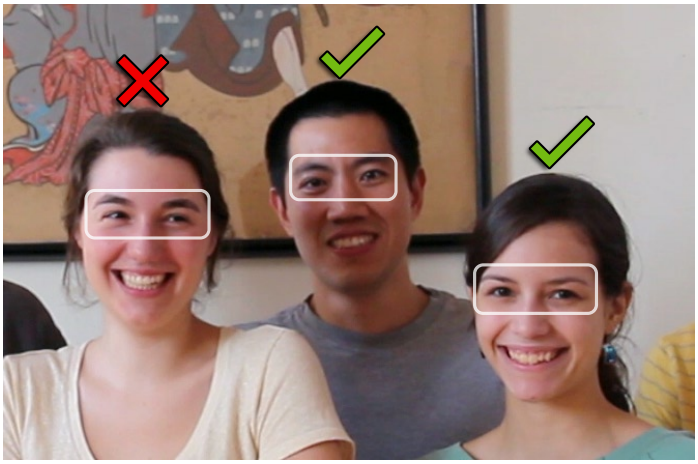


Figure 4.1: Gaze locking. We propose the idea of sensing eye contact directly from an image in a passive, appearance-based manner. The main idea is to focus on gaze *locking* (a binary problem) rather than gaze tracking (a continuous problem) and exploit the special appearance of direct eye gaze. Our approach can be used to facilitate a wide range of applications.

Kriegman, and Ahuja 2002] are highly accurate, they suffer from several limitations that restrict their practical use. They generally work only over short distances (often 80 cm or less) [Baluja and Pomerleau 1994; Beymer and Flickner 2003; Hansen and Pece 2005; Morimoto, Amir, and Flickner 2002; Tan, Kriegman, and Ahuja 2002] or with direct head poses [Baluja and Pomerleau 1994; Hansen and Pece 2005; Tan, Kriegman, and Ahuja 2002], or require active infrared illumination [Baluja and Pomerleau 1994; Beymer and Flickner 2003; Morimoto, Amir, and Flickner 2002; Tan, Kriegman, and Ahuja 2002], intrusive equipment (such as head-mounted cameras), or extensive calibration [Beymer and Flickner 2003; Morimoto, Amir, and Flickner 2002]. One exception is Shell et al.'s system [Shell, Vertegaal, Cheng, Skaburskis, Sohn, Stewart, Aoudeh, and Dickie 2004], which does in fact sense eye contact directly, but also requires active illumination. To overcome these limitations, we show the following in this chapter.

Gaze Locking Contributions

Gaze Locking

We propose the idea of sensing eye contact directly from an image in a passive, appearance-based manner (Figure 4.1). The main idea is to focus on gaze *locking* (a binary problem) rather than gaze tracking (a continuous problem) and exploit the special appearance of direct eye gaze. In addition to being passive (no special illumination or hardware required), our approach is non-intrusive, calibration-free, and robust to distance and head pose. Like Shell et al.'s active method [Shell, Vertegaal, Cheng, Skaburskis, Sohn, Stewart, Aoudeh, and Dickie 2004], it can be used to allow humans to interact with computers, devices, and other objects just by looking at them.

Sample Detector

As a proof of concept, we demonstrate that even a simple and lightweight gaze locking system can yield accurate and robust results. Our sample detector uses very basic features—the eye area's pixel intensities—yet achieves a Matthews correlation coefficient (a measure of accuracy for binary classifiers) of over 0.83 at long distances (up to 18 m) and large pose variations (up to $\pm 30^\circ$ of head yaw rotation) without requiring calibration. This equates to a 92% accuracy on our training data set. It runs at over 20 FPS on a computer with an Intel Core i5-3470 processor, 8 GB of RAM, and an NVIDIA GeForce GTX 660M graphics card. A more advanced classifier could be used to improve accuracy even further.

Human Performance Evaluation

We performed a study to see how accurate people are at sensing eye contact and found several interesting results. For example, we found that people achieve MCCs of over 0.2 at distances of 18 m, and that their accuracy decreases roughly linearly over distance regardless of others' (horizontal) head orientations. We also found that people are often more accurate when they can only see one of the other person's eyes.

Gaze Data Set

To facilitate our human study and provide training data for our sample detector, we created a gaze data set of 56 people and 5,880 images, available at `http://www.cs.columbia.edu/CAVE/databases/columbia_gaze/`. It has more images and fixed gaze targets than any other publicly available gaze data set. To ensure robustness, our data set spans a variety of parameters: 5 head poses and 21 gaze directions per head pose. Our subjects were ethnically diverse and 21 of them wore glasses. We use our data set for gaze locking purposes, but it can serve as a very large resource for gaze tracking purposes as well.

Demonstration of Applications

Lastly, we show a few applications that gaze locking facilitates. First, since camera modules are becoming increasingly small and inexpensive to produce, we can allow any device to respond to eye contact by embedding a camera that serves as an eye contact sensor inside it. Our technique can thus serve as a backbone for allowing humans to interact with computers, devices, and other objects simply by looking at them. In addition, our method is passive and hence can be applied to any existing image. Therefore, it can be used to sort

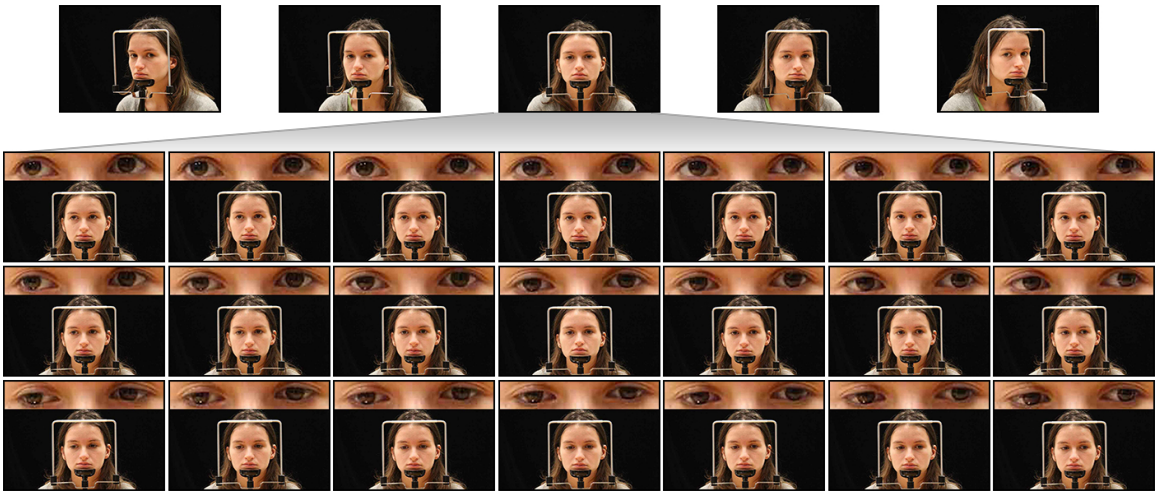


Figure 4.2: Sample Columbia Gaze Data Set images. Our gaze data set includes 56 subjects and five different head poses (shown on top): $0°$, $\pm15°$, and $\pm30°$ horizontally. For each subject and head pose, there are 21 different gaze directions (shown on bottom for the $0°$ head pose): the combinations of seven horizontal ones ($0°$, $\pm5°$, $\pm10°$, $\pm15°$) and three vertical ones ($0°$, $\pm10°$). For each of these, we also show a cropped area of the eye region.

images on the web and on personal computers by their degree of eye contact, improving image search. Finally, we can incorporate a gaze trigger in cameras to capture group photos exactly when everyone in the group is looking straight back.

4.1 Gaze Locking in People

People seem to have an uncanny ability to tell when others are looking at them. In this section, we describe a two-part experiment that we performed to show how accurate people really are. First, we created a gaze data set, then we asked a set of "players" to determine which of those images are gaze locking and which ones are not. Our experiment revealed some very interesting trends in human vision, and we used those to guide the design of our gaze locking approach.

Creating a Gaze Data Set

Data Set Statistics

Our data set contains a total of 5,880 high-resolution images of 56 different people (32 male, 24 female), and each image has a resolution of 5,184×3,456 pixels. 21 of our subjects were Asian, 19 were White, 8 were South Asian, 7 were Black, and 4 were Hispanic or Latino. Our subjects ranged from 18 to 36 years of age, and 21 of them wore prescription glasses.

As shown in Figure 4.2, for each subject, we acquired images for each combination of five horizontal head poses (0°, ±15°, ±30°), seven horizontal gaze directions (0°, ±5°, ±10°, ±15°), and three vertical gaze directions (0°, ±10°). Note that this means we collected five gaze locking images (0° vertical and horizontal gaze direction) for each subject, one for each head pose. Figure 4.3 compares our gaze data set with data sets recently made by McMurrough et al. [McMurrough, Metsis, Rich, and Makedon 2012], Ponz et al. [Ponz, Villanueva, and Cabeza 2012], and Weidenbacher et al. [Weidenbacher, Layher, Strauss, and Neumann 2007] for gaze tracking.

Collection Procedure

We recorded each image with a Canon EOS Rebel T3i camera and a Canon EF-S 18–135 mm IS f/3.5–5.6 zoom lens. As shown in Figure 4.4, subjects were seated in a fixed location in front of a black background, and a grid of dots was attached to a wall in front of them. The dots were placed in 5° increments horizontally and 10° increments vertically. There were five camera positions marked on the floor (one for each head pose), and each position was 2 m from the subject. The dots were organized in such a way that each camera

	McMurrough et al.	Gi4E	Weidenbacher et al.	Our Data Set
# Subjects:	20	103	20	56
# Fixed Gaze Targets per Subject per Head Pose:	16	12	2-9	21
# Fixed Head Poses:	1	N/A	19	5
Head Pose Calibration?	N/A	No	Yes	Yes
Resolution (px):	768×480	800×600	1600×1200	5184×3456
Total # Images:	N/A	1,236	2,220	5,880

Figure 4.3: Gaze data set comparison. A comparison of our gaze data set with ones recently made for gaze tracking. McMurrough et al.'s data set is video-based and includes precise head pose measurements rather than simply calibrating head pose. The Gi4E data set does not stabilize subjects' head pose. Weidenbacher et al.'s data set offers a wide variety of fixed head poses, but many have only two corresponding gaze directions.

position had a corresponding 7×3 grid.

The subjects used a height-adjustable chin rest to stabilize their face and position their eyes 70 cm above the floor. The camera was placed at eye height, as was the center row of dots. For each subject and head pose (camera position), we took three to six images of the subject gazing (in a raster scan fashion) at each dot of the pose's corresponding grid of dots. To ensure the subject was in focus, not blinking, and looking in the correct direction, we viewed each image at full resolution afterwards and kept the best one from each set of three or six.

Human Accuracy

Experimental Setup

After creating our gaze data set, we asked 52 "players" (27 male, 25 female) to play a computer-based quiz to determine which of those images are gaze locking and which of

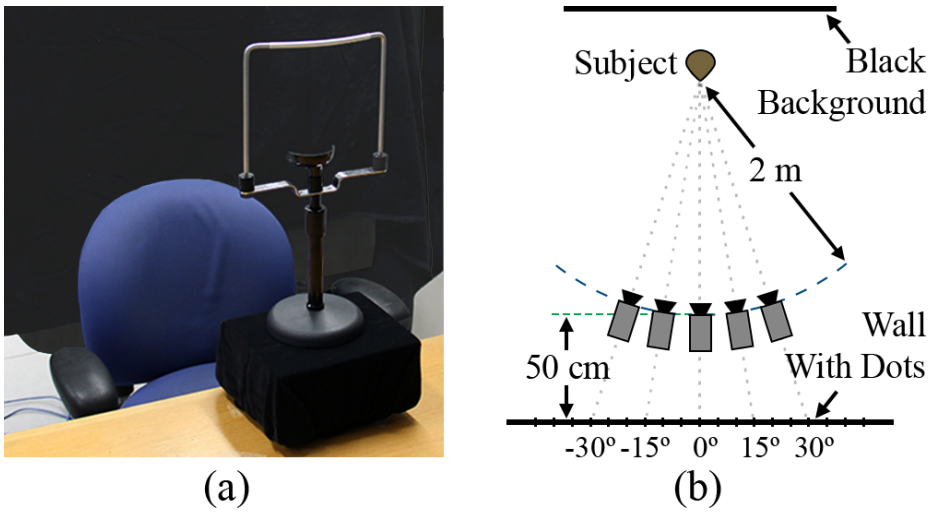


Figure 4.4: Setup for image capture. (a) Subjects were seated in front of a black background and used a chin rest to stabilize their face. (b) We captured images from five different camera positions asynchronously. Each position represented a (horizontal) head pose. The subjects focused on a grid of dots placed on the wall behind each camera location.

those are not. The players were all paid volunteers and were mostly university students. We asked the players to state whether or not the subject in each image was looking directly at him or her, a simple yes/no response. Each gaze-locking image was seen by an average of 8.8 players and each non-gaze-locking image was seen by an average of 3.96 players. Each player participated in one 40-minute session, viewing 440 images in the process.

The players viewed our images on a computer monitor, so we needed the subjects to appear at the same resolution as they would if seen in person for the results to be accurate. In addition, each image was captured at a distance of 2 m from the subject, so we created four more copies of each image to serve as a proxy for distances of 6 m, 10 m, 14 m, and 18 m. Section 4.2 describes how we scaled the images—taking the acuity of the human eye into account—to solve both of these problems. We did not find a statistical difference in people's accuracy when we used a small sample of "true" 6-18 m images instead of our scaled images.

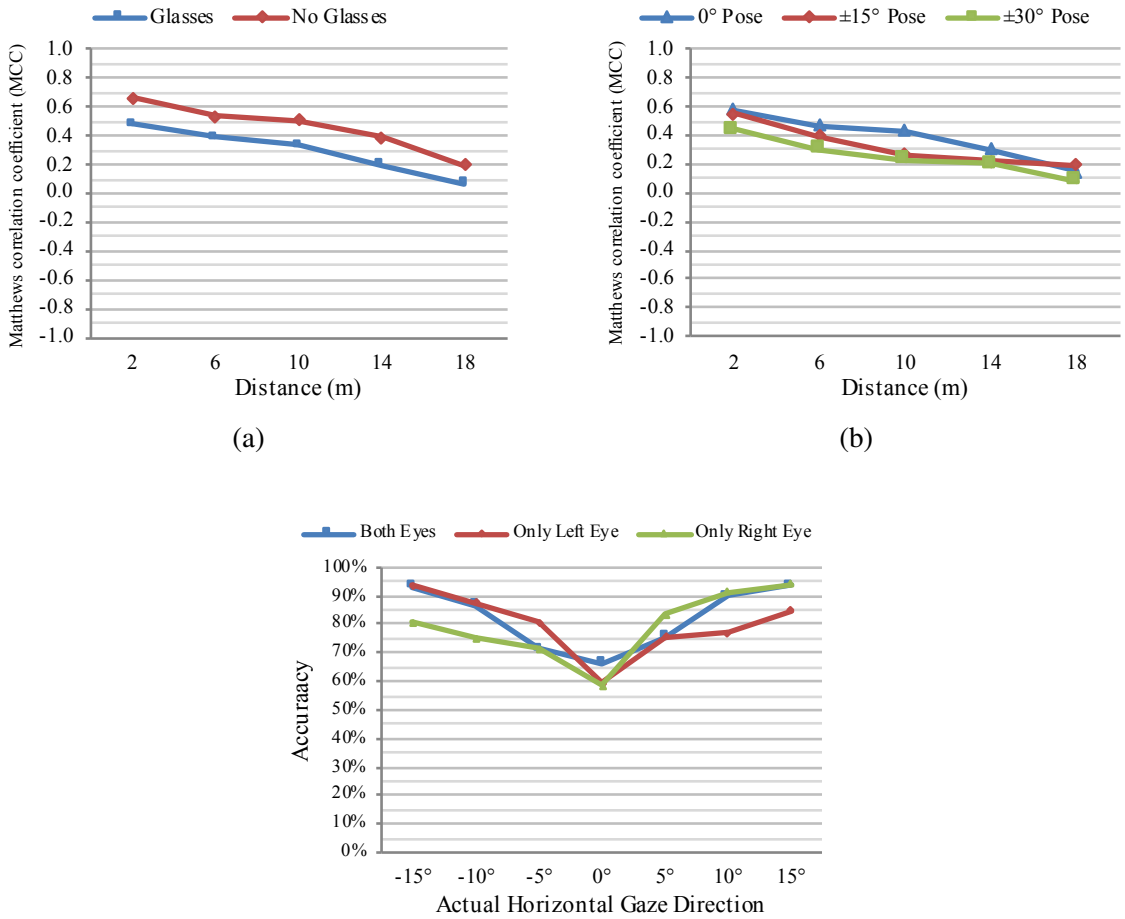


Figure 4.5: Gaze locking in people. (a) People are relatively accurate at sensing eye contact, even when the person gazing (i.e., the gazer) is wearing prescription glasses. At distances of 18 m, gazees still achieve MCCs of over 0.2 if the gazer is not wearing glasses. Here, the gazer is at a frontal (0°) head pose. (b) The gazee's accuracy decreases roughly linearly over distance regardless of the gazer's (horizontal) head pose. Head poses that are more off-center (such as ±30°) have slightly lower MCCs. (c) The gazees are least accurate when the gazer is actually looking at them (the 0° case)—that is, the false negative rate is higher than the false positive rate. Interestingly, if the gazer is looking away, the gazee is more accurate when he or she can only see one of the gazer's eyes (the blue line is not strictly above the red and green lines). Each accuracy measurement was calculated over all five distances and head poses. Here, we use percentage accuracies instead of MCCs because each horizontal gaze direction besides 0° is always non-gaze locking by definition and the MCC is not well-defined when only one class of data is used.

Human Accuracy Results

Figure 4.5 highlights some of our observations. We use the Matthews correlation coefficient (MCC) [Matthews 1975] to represent accuracy in Figure 4.5(a) and Figure 4.5(b) since it is widely used in machine learning for assessing binary classification performance with uneven class sizes. An MCC of 1.0 represents perfect classification, an MCC of -1.0 represents completely incorrect classification, and an MCC of 0.0 represents classification that is no better than random guessing. The MCC is not well-defined when only one class of data is used, so we use percentage accuracies in Figure 4.5(c).

In Figure 4.5(a), we find that humans are indeed rather adept at determining when others are looking at them. At distances of 18 m, their MCC can still surpass 0.2. Moreover, even though the size of someone else's face decreases quadratically over distance, humans' gaze locking accuracy decreases only linearly with distance.

In Figure 4.5(b), we see that humans' accuracy is largely maintained across different head poses, even extreme ones such as $\pm 30^\circ$ to the side. Shechtman et al. [Shechtman, Riordan-Eva, and Hardigan 2005] find that a 50° ocular duction (i.e., eye movement) is nearly impossible for many age groups, and we found during our experiments that people are uncomfortable moving their eyes 30° to the side. Hence, eye contact from even a $\pm 30^\circ$ head pose is unlikely to happen in everyday life.

Lastly, Figure 4.5(c) shows the results from an extended test that we performed with 10 players. In this test, the players viewed images whose left or right half was cropped off, showing only one of the subject's eyes, in addition to non-cropped images. Interestingly, we found that the players are often more accurate when they can only see one of the other

person's eyes. This is the case when the subject is looking away and the visible eye is the one that is looking more off-center.

4.2 Scaling Columbia Gaze Data Set Images

In the experiment described above, our players (study participants) viewed our images in front of a computer screen, but we needed the subjects in the images to appear as they would in person for the results to be accurate. We also needed to represent a variety of distances (2 m to 18 m) properly for both the human and sample detector experiments. Hence, we took the parameters of our camera, computer monitor, and even the acuity of the human eye into account to scale the images accordingly and display them at the proper resolution. The calculations are described here.

Human Test

An "eye pixel" is the smallest area of the human fovea that can distinguish a point or a line pair [Blackwell 1946; Curcio, Sloan, Kalina, and Hendrickson 1990]. In our human experiments, if the player were to view the subject directly from a distance of d_o, the subject's face would subtend E of the player's eye's pixels in width, where:

$$E = \frac{w}{2d_o \tan(\theta_e/2)}. \tag{4.1}$$

w is the width of the subject's face (usually $\approx$14 cm). θ_e is the angular resolution of the human eye fovea, and is roughly 0.3 arc-minutes (or 0.005° per eye pixel) [Blackwell 1946;

Curcio, Sloan, Kalina, and Hendrickson 1990].

When the subject is captured by a camera instead, his or her face subtends C camera pixels in width, where:

$$C = \frac{P_c w}{w_c(u/f - 1)}. \tag{4.2}$$

P_c is the camera's horizontal pixel count, w_c is the width of the image sensor, f is the camera's focal length, and u is the distance from the subject to the camera.

Then, if the player views the captured image on a screen, the subject's face would subtend S eye pixels in width:

$$S = \frac{\alpha C w_s}{2 d_s P_s \tan(\theta_e/2)}, \quad \text{where} \quad \alpha \leq 1. \tag{4.3}$$

α is the factor by which the image dimensions are scaled on the screen (1 represents 100%), P_s is the screen's horizontal pixel count, w_s is the screen's width, and d_s is the distance from the player to the screen.

For the player to view the subject on the screen without a loss in resolution compared to seeing the subject in person, both C and S must be greater than or equal to E. This was true in our configuration, in which u = 2 m, f = 85 mm, P_c = 5184 px, w_c = 22.3 mm, P_s = 2560 px, d_s = 664 mm, D = 508 mm, and the corresponding (d_o, α) pairs (i.e., the scale factors we used to represent each distance) were (2 m, 19.5%), (6 m, 6.5%), (10 m, 3.9%), (14 m, 2.8%), and (18 m, 2.2%).

Sample Detector Experiments

When testing the accuracy of our sample detector, we wanted the detector to "see" each image as if they were being viewed by a person at the appropriate distance d_o. That is, the subject's face in each image should be of the same resolution that the human retina would see it at a distance of d_o. Hence, we can simply downsample our data set's images to a resolution of E via Equation 4.1 to get versions of them that correspond to different distances d_o.

4.3 Sample Gaze Locking Detector

Here, we show a simple, lightweight detector design that is nonetheless accurate and robust. It runs at over 20 FPS on a computer with an Intel Core i5-3470 processor, 8 GB of RAM, and an NVIDIA GeForce GTX 660M graphics card.

Given an image, the detector outputs binary decisions that indicate whether each face in the image is gaze locking or not. It is composed of three broad phases, shown in Figure 4.6 and described below.

Pre-Processing Phase

In the first phase, we locate the eyes in an image and transform them into a standard coordinate frame. We find the eyes by taking the eye corner locations output from a commercial face and fiducial point detector [Omron 2012]. We rectify each eye via an affine transformation to remove the influence of head pose. Figure 4.7 shows several examples of rectified features.

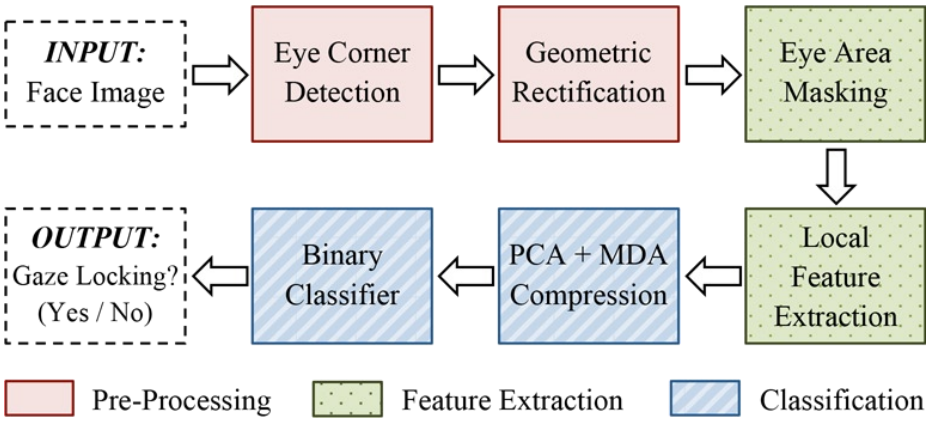
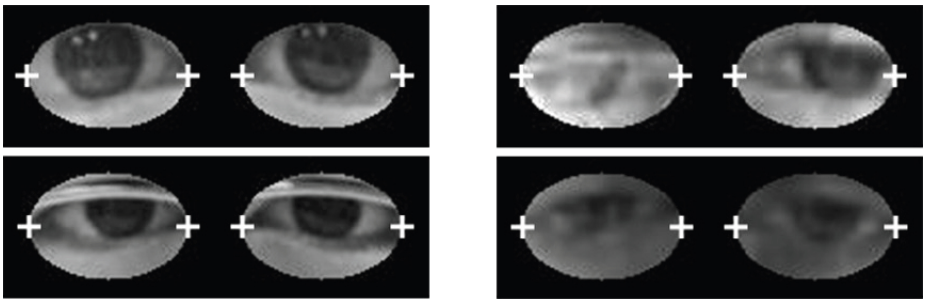


Figure 4.6: Gaze locking detector pipeline. Our gaze locking detector is comprised of three broad phases, shown here in different colors. In the first phase, we locate the eyes in an image and transform them into a standard coordinate frame. In the second phase, we mask out the eyes' surroundings and assemble pixel-wise features from the eyes' appearance. Finally, we project these features into a low-dimensional space, then feed them into a binary classifier to determine whether the face is gaze locking or not.



(a) (b)

Figure 4.7: Rectified eye features and gaze locking failure cases. (a) Examples of rectified and masked features. Each eye has been transformed to a 48×36 px coordinate frame. The crosshairs signify eye corners detected in the first phase. We mask each eye with a fixed-size ellipse whose shape was optimized offline for accuracy. (b) Two failure cases: strong highlights on glasses (top) and low contrast (bottom).

Feature Extraction Phase

The most difficult part of using the eyes' appearance for classification purposes is the inherent variance in the eyes' appearance. Both the eyes' shape and degree of openness significantly affect their appearance, even after performing the affine transformation in the first phase. Training our detector with a large number samples helps account for this, but we take the additional step of masking out the areas around the eyes to remove the influence of their variances in appearance.

Our mask (Figure 4.7) is a fixed-size ellipse whose major axis lies on the line segment connecting the two eye corners. Choosing the size is nontrivial: a larger ellipse reveals more of the eye's surroundings and more information about gaze, but a smaller ellipse is more robust to noise from the surroundings. We used a brute-force search of all possible major and minor axis lengths offline to choose the best size. We chose the values that achieved the best accuracy in our set of training data, which is separate from our testing data.

After applying the mask, we concatenate the remaining pixels' intensity values into a high-dimensional feature vector, then normalize the feature vector to unit magnitude. This unit-magnitude feature vector is our final representation of the eyes' appearance.

Classification Phase

In the final phase, we project the high-dimensional feature vector onto a low-dimensional space via principal component analysis (PCA) [Turk and Pentland 1991] and multiple discriminant analysis (MDA) [Duda, Hart, and Stork 2001], then feed the projected vector

into a support vector machine (SVM) [Chang and Lin 2011] that we trained offline. The SVM decides whether the face is gaze locking or not. Since gaze locking is a binary classification problem rather than a continuous one, it is more robust to noise and requires fewer training samples. The Binary Classifier subsection describes the training process.

PCA + MDA Compression

Dimensionality reduction is a common task in appearance matching. It boosts classification speed, removes redundancies in the representations of features, avoids over-fitting, and reduces the effects of noise on classification. Hence, we use PCA to compress our feature vector to roughly 200 dimensions. Afterwards, we employ MDA to form a highly discriminative subspace, compressing our feature vectors even more. We find that a six-dimensional subspace, used to separate seven distinct classes of data, yields the highest accuracy. One class corresponds to gaze locking images and the rest correspond to non-gaze-locking images. This is likely because our training data set comprises seven horizontal gaze directions, one of them gaze locking (0°) and the rest non-gaze-locking.

Binary Classifier

In our sample detector, we use a linear SVM classifier [Chang and Lin 2011] with default parameters (which includes a radial basis function kernel) to output our final binary decision. Even though we use an SVM, any binary classifier (e.g., LDA or neural networks) would work. The kernel allows input features similar to our positive training samples to be "lifted airborne," so to speak, separating them from ones near our negative samples that are still "on the ground." As was the case with the data set statistics described earlier,

the number of positive and negative training samples we had was highly unbalanced (280 gaze locking images and 5,600 non-gaze-locking images), so we randomly perturbed our training data to generate 5,000 additional gaze locking samples and 15,000 additional non-gaze-locking samples. We accomplished this by making small, random adjustments to the resolution and detected eye corner positions of our training images.

4.4 Experiments

We tested our gaze locking detector via leave-one-out cross-validation on a modified version of our gaze data set. The original data set, described earlier in the Creating a Gaze Data Set subsection, comprises five head poses and 21 gaze directions but was captured with a high-resolution camera from a distance of 2 m. The modified data set comprises five downsampled copies of each of the original data set's 5,880 images, where the resolution of each copy matches that seen by the human retina at distances of 2 m, 6 m, 10 m, 14 m, and 18 m. Equation 4.1 in Appendex A describes how we downsampled the images. Note that even the 2 m image is downsampled by a factor of 58.0%. Our supplementary video shows how our detector performs on raw footage from webcams and iPad video feeds.

Comparison with Human Vision

Figure 4.8 shows our sample detector's performance and compares it with human vision's performance (from Figure 4.5). We again use the Matthews correlation coefficient (MCC) [Matthews 1975] to represent accuracy since it is widely used in machine learning for assessing binary classification performance with uneven class sizes. Although our

detector uses a very standard set of tools, it achieves an MCC of over 0.83 at a distance of 18 m, significantly outperforming the 0.15 MCC of human vision (Figure 4.8(a)). This high accuracy over long distances is a result of using very low-resolution feature vectors. As we see in Figure 4.8(b), this accuracy is maintained across different horizontal head poses, even fairly extreme ones such as $\pm 30^\circ$.

Figure 4.8(c) shows our detector's accuracy with respect to a person's actual gaze direction. As with human vision, our detector is least accurate when a person is looking at or very near the camera (i.e., at the borderline between gaze locking and "almost gaze locking"). However, even in this case, our detector is much more accurate than human vision (86% vs. 67%). We use percentage accuracies instead of MCCs here because each horizontal gaze direction besides 0° is always non-gaze locking by definition and the MCC is not well-defined when only one class of data is used.

Comparison with an Active System

Here, we compare our sample detector's accuracy with that of the eyebox2 [eyebox2 2007], a leading commercial implementation of Shell et al.'s active approach to eye contact detection. Recall that our approach is completely passive and hence does not use active illumination or special hardware like the eyebox2 does. The eyebox2 is specified to work best at a range of 5–10 m in Normal mode and 1.3–3.3 m in Close Range mode, so we asked six people to sit (indoors using a chin rest) 6 m in front of it in Normal mode and 2 m in front of it in Close Range mode. They stared at the eyebox2 and six dots placed horizontally around it (at $\pm 5^\circ$, $\pm 10^\circ$, and $\pm 15^\circ$) for ten seconds apiece. To analyze the eyebox2's

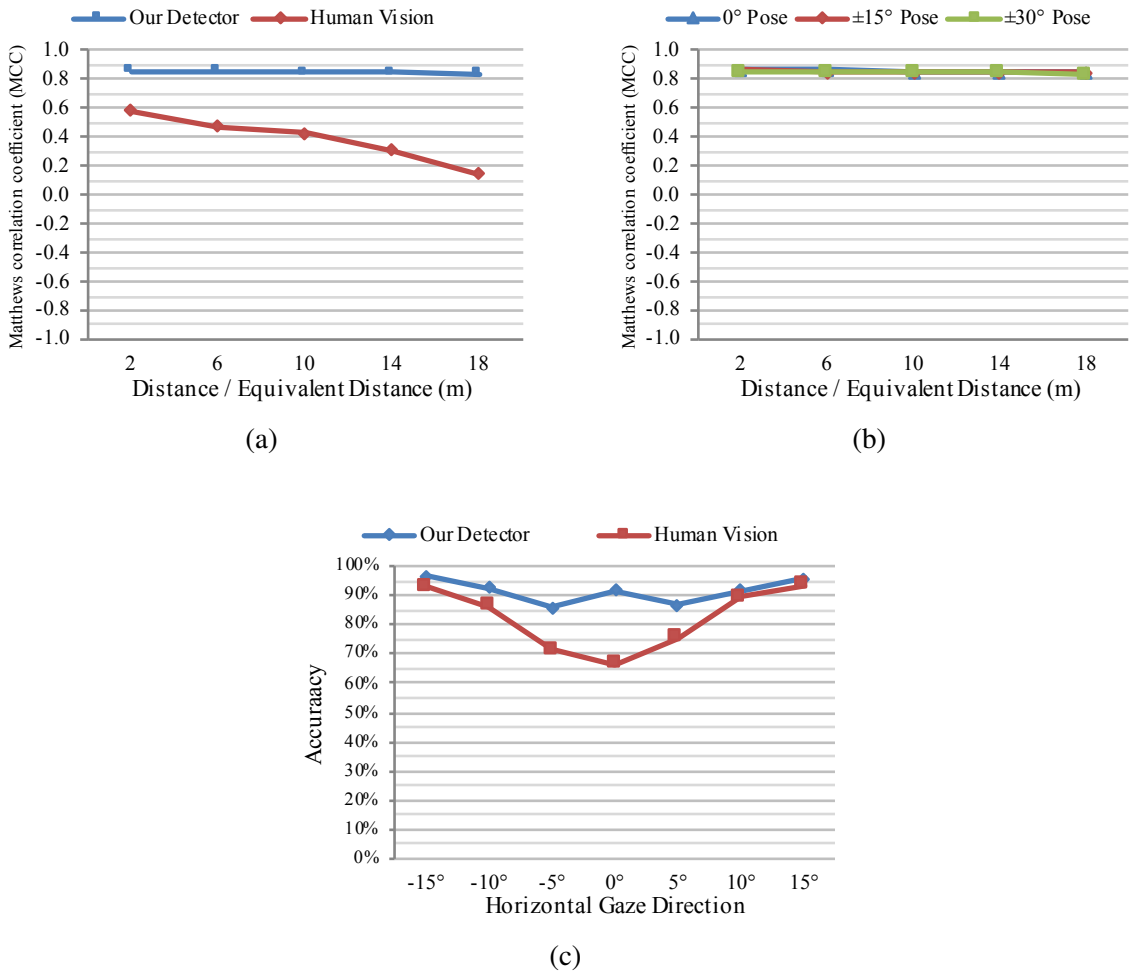


Figure 4.8: Gaze locking detector performance. We downsampled our detector's test images to match the resolution seen by the human fovea at the respective distances. (a) Our detector achieves MCCs of over 0.83 at a distance of 18 m, significantly outperforming humans' accuracy. The detector's accuracy is fairly constant over distance because our method uses very low resolution features. (b) Our detector's accuracy is also fairly constant over a variety of (horizontal) head poses. (c) As with human vision, our detector's accuracy is worst when people are looking at or very close to the camera. Our detector significantly outperforms human vision nonetheless. We use percentage accuracies here because the MCC is not well-defined when only one class of data is used.

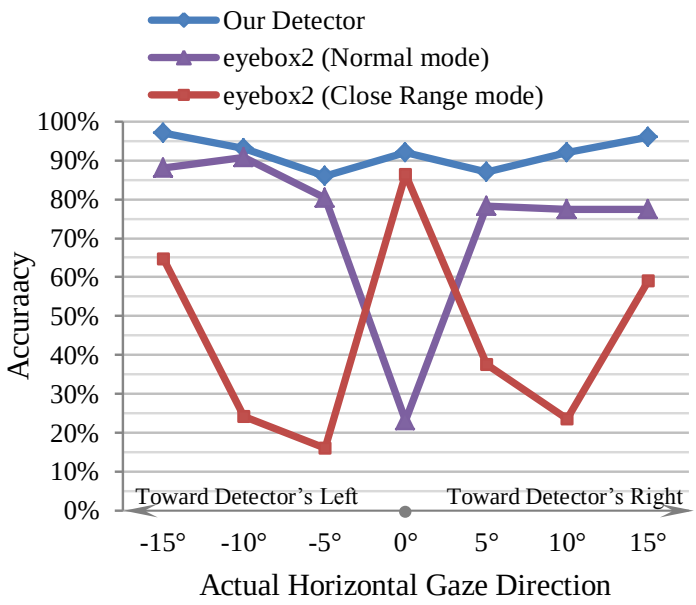


Figure 4.9: Comparison with an active system. A comparison of our sample detector with an eyebox2, which implements Shell et al.'s active approach to eye contact detection, in both Normal (6 m) and Close Range (2 m) modes. Though passive, our detector is more accurate than the eyebox2. The eyebox2's Normal mode seems to be tuned toward reducing false positives, and its Close Range mode seems to be tuned toward reducing false negatives.

accuracy, we measured the proportion of time the eyebox2 claimed they were making eye contact.

Figure 4.9 shows the results. Our sample detector is more accurate than the eyebox2 regardless of the actual gaze direction. In Normal mode, the eyebox2 seems tuned toward reducing false positives—although we adjusted its illuminator position, threshold setting, and focus setting as the manual instructed, its output was very jittery in the gaze locking case. We also found its range for reliable tracking to be around 5–7 m, and that it does not work well for people with glasses (the participants represented in Figure 4.9 did not wear glasses). In Close Range mode, the eyebox2 seems to be tuned toward reducing false negatives—it usually claims that people are looking at it unless they are looking at least 15° away. Our gaze locking approach works for a greater range of distances (at least 2–18 m)

without separate modes of operation, is more robust to eyeglasses, and can be applied to any image, including existing images.

Failure Cases

As with all appearance-based recognition systems, our approach can be prone to errors when a feature's visual appearance (in our case, that of the eyes) differs significantly from that of average features. For example, even though 22 of 56 subjects in our training data set wore glasses, our detector may not work well for all types of glasses over all head poses, given the large variety. This, however, is also true for active techniques that rely on reflections. Our approach is also prone to errors when the eyes are severely occluded (e.g., if a person's hair blocks an eye), when the illumination is extreme (e.g., strong highlights or profile illumination), or when there is very low contrast in the image. Figure 4.7(b) highlights two of our sample detector's failure cases.

4.5 Applications of Gaze Locking

We now demonstrate four of the applications that gaze locking facilitates. For each of these applications, we recorded video feeds from the respective devices (iPads, webcams, and a DSLR camera) and ran our detector on them offline.

Human–Object Interaction

Cameras are becoming increasingly small and inexpensive to produce. By embedding cameras in everyday devices and objects, the devices and objects can be selected or activated

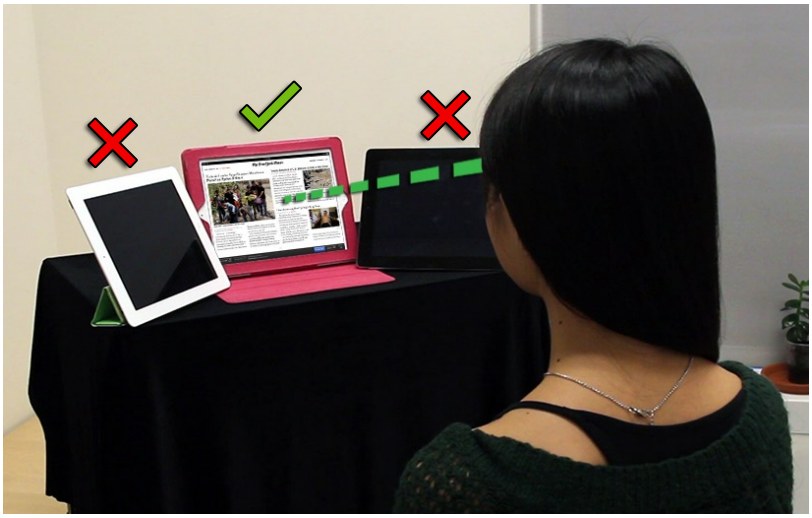


Figure 4.10: Human–object interaction. Our gaze locking approach allows people to interact with objects just by looking at them. In this proof of concept, we process the videos from the embedded cameras of three iPads to sense when the iPads are being looked at. Here, the woman is looking at the iPad in the middle. Since the iPads' cameras are on their extreme left, she was instructed to look at the iPads' left halves.

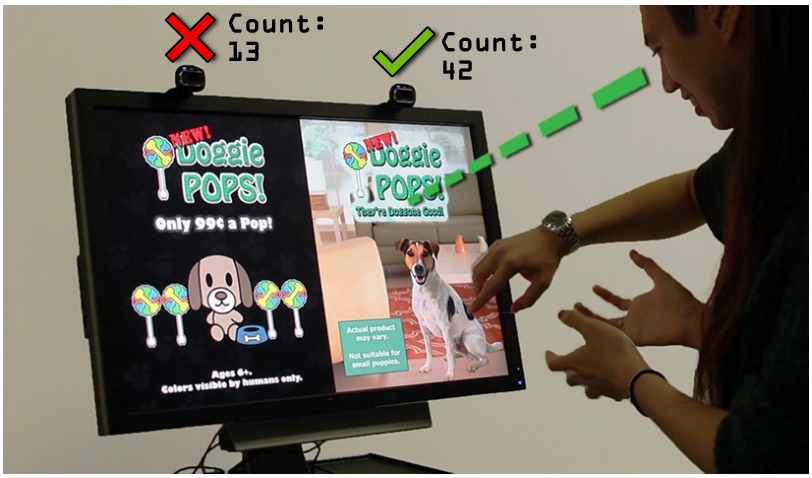


Figure 4.11: User analytics. Two ordinary webcams are placed above two ads for the same product. By counting the number of times each advertisement is viewed, we can gauge which one is more effective. The counts incremented when the viewers looked at the ads' top halves.

simply by looking at them.

As an example, Figure 4.10 shows a proof of concept system that we created with 3rd generation iPads. We process the videos from the iPads' built-in cameras (which have a 640×480 px resolution) to sense when they are looked at, then display relevant content such

as news headlines or reading lists. As another example, a museum exhibit or department store item could be rigged with a small camera to inform passersby about them when they look at them.

Some smartphones (e.g., the Samsung Galaxy S III and the Samsung Galaxy S4) already include "smart pause" and "smart scroll" features to pause videos by looking away from the phone and scroll documents by looking up and down. However, we found that both features on the Galaxy S4 seem to work reliably only when a user moves his or her entire head, although the system also responded sometimes to large eye motions alone. Our technique can distinguish eye contact from a subtle $\pm 5^\circ$ gaze away, and it works over long distances.

User Analytics

Several commercial systems [eyebox2 2007; EyeTech 2013] embed cameras in product displays or advertisements as a means of measuring consumer attention, but these systems employ active infrared illumination. As Figure 4.11 shows, our method offers a completely passive alternative that is robust to distance and does not require special hardware.

As with the commercial systems, our method has a reasonably sized tolerance for what it considers gaze locking, allowing it to work for cameras placed adjacent to regions of interest as well. Our detector was trained to distinguish between 0° and $\pm 5^\circ$ horizontally and between 0° and $\pm 10^\circ$ vertically, so we estimate its tolerance to be roughly 2.5° in either direction horizontally and 5° in either direction vertically. This corresponds to a 8.7×17.5 cm target at a distance of 1 m and a 43.7×87.5 cm target at a distance of 5 m.

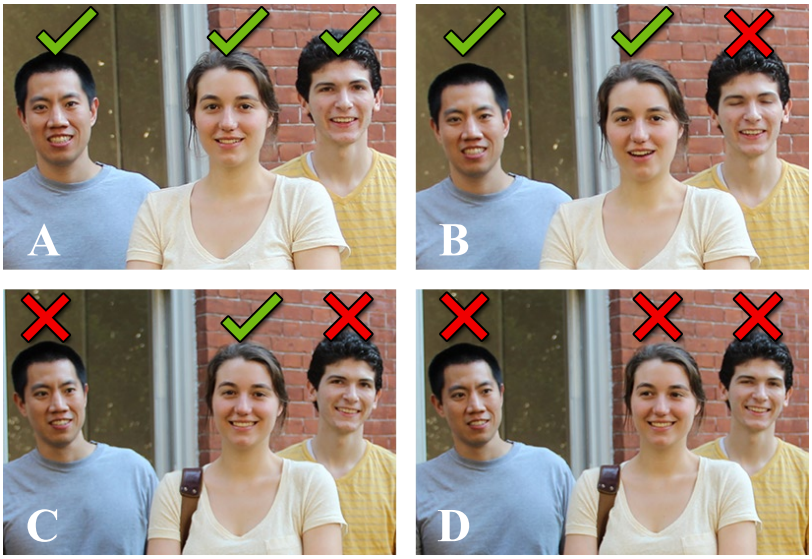


Figure 4.12: Image search filter. Our approach is completely appearance-based and can be applied to any image, including existing images such as ones from the Web. Hence, we can sort these images (A–D) by degree of eye contact to quickly find one where everyone is looking at the camera. These are actual decisions made by our detector.

Image Filtering

Unlike active methods, our method can be used to detect eye contact in existing images such as ones from the Internet. There are billions of images on the Internet, and over 300 million photos are uploaded to Facebook alone each day [Kiss 2012]. Hence, as Figure 4.12 shows, we can sort and filter photos by degree of eye contact with our method to improve image search.

Gaze-Triggered Photography

With today's cameras, taking a group photo can be difficult since everyone must be looking at the camera at the right moment. With our technology, however, cameras can incorporate a gaze trigger that works as follows: the photographer would initiate the function, then join the group as the camera waits to see another face (the photographer's) enter the frame. As

Figure 4.13: Gaze-triggered photography. By incorporating a gaze locking detector in a consumer-level camera, the camera could automatically take a picture when the entire group is looking straight back, allowing the photographer to join the group and still capture a perfect photo. Our accompanying video shows our detector's output on the camera's feed.

soon as this is detected, the camera would take a picture when the entire group is looking straight back. Figure 4.13 and our supplementary video demonstrate this concept.

Already, many consumer-level cameras (e.g., the Sony Cyber-shot W650) feature an anti-blink function that helps users capture photos when subjects are not blinking. Other cameras (e.g., the Canon PowerShot XS160 IS) also include face self-timers that release the shutter only when an additional face (the photographer's) enters the frame. By sensing eye contact instead of simply sensing blinking or the presence of faces, our technology can make cameras aware of when people are actually looking straight back.

4.6 Discussion

In this work, we have created a passive approach for sensing eye contact from a live camera or an existing still image or video recording and demonstrated several of the applications that it facilitates, such as human–object interaction and gaze-triggered photography. We

also performed a study on how accurately humans can perform the same task, finding several interesting results. Lastly, we created a large gaze data set. Unlike existing gaze tracking approaches, our approach exploits the special appearance of direct eye gaze, making it largely robust to distance and pose, even though it is passive, non-intrusive, and calibration-free. Furthermore, it does not require any special hardware.

Section 7.2 describes what we feel is the future of gaze locking.

Chapter 5

Optimizing Touchscreen Keyboards for Gesture Typing

This chapter explores how to modify existing gesture typing keyboards to make typing on small devices much faster and more error-free than is currently possible. It addresses a central usability problem pertaining to mobile devices: that of entering text, which is inevitable in the course of using mobile devices. It also makes it as painless as possible for users to type on small devices when their inputs cannot be anticipated by the system and their inputs are complex. Touchscreen typing in particular is slow and prone to errors and typos.

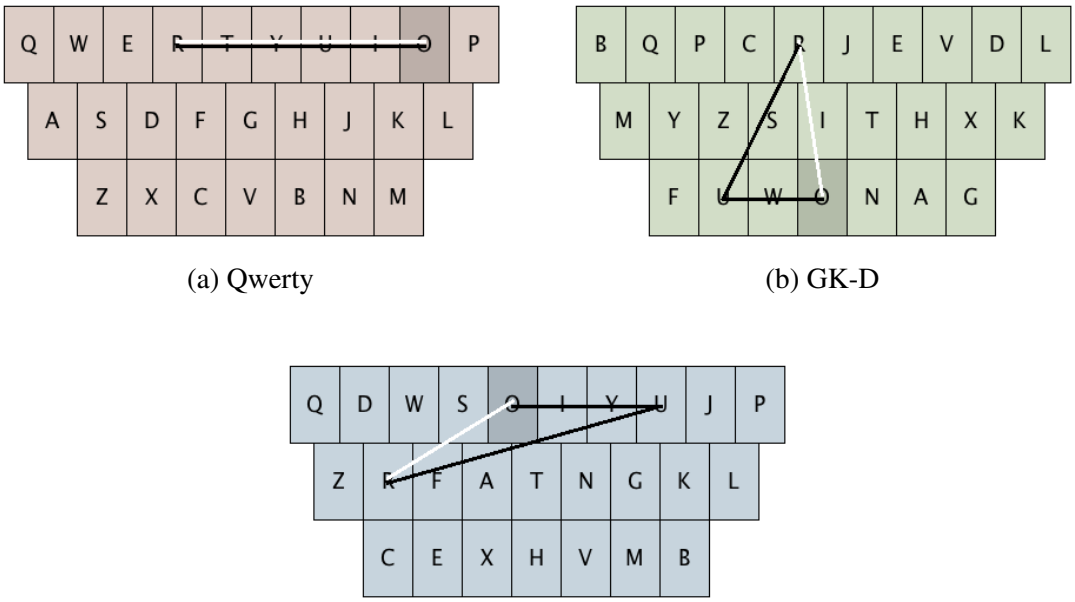
Specifically, we explore modifying Qwerty, the standard keyboard layout, to make word gestures shorter and more distinct without making users have to learn how to type all over again. The impact of doing so may be enormous: the amount of time humanity collectively spends typing e-mail alone on mobile devices is over 120 millennia per day[1].

[1]We can see this with some back-of-the-envelope calculations. Using the conservative assumption that at least 10% of e-mail is sent from mobile devices and the facts that as of 2018 people send 281 billion e-mails globally each day [*Email Statistics Report, 2018-2022*], the average e-mail length is 47 words [*You Probably Write a Novel's Worth of Email Every Year*], and the average smartphone typing speed is 20 words per minute [MacKenzie, Zhang, and Soukoreff 1999], we can calculate that humanity collectively spends over 120,000 years each day typing e-mails on mobile devices.

5.1 Gesture Typing Advantages and Pitfalls

Gesture typing offers several advantages over touch typing (tapping): it supports a gradual and seamless transition from visually guided tracing to recall-based gesturing, allows users to approximate words with gestures rather than tapping each key exactly, and mitigates one major problem plaguing regular touchscreen typing: the lack of tactile feedback. Since its invention in the early 2000's [Zhai and Kristensson 2003], gesture typing has gained large-scale adoption on mobile devices and can be found on all major mobile computing platforms in products such as ShapeWriter, Swype, SwiftKey, SlideIT, TouchPal, and Google Keyboard.

Despite these benefits, gesture typing suffers from an inherent problem: highly ambiguous word gestures. Bi et al. [Bi, Azenkot, Partridge, and Zhai 2013b] showed that the error rate for gesture typing is approximately 5–10% higher than for touch typing. This problem occurs because when gesture typing, the input finger must inevitably cross unintended letters before reaching the intended one. The Qwerty layout itself further exacerbates this problem. Because common vowels such as 'u,' 'i,' and 'o' are arranged together on Qwerty, many pairs of words (such as "or" and "our") have identical gestures, and many others (such as "but" and "bit") have very similar gestures. Figure 1(a) shows the gestures for "or" and "our" — their superstrings "for" and "four" also have identical gestures. In fact, our analysis over a 40,000-word lexicon showed that 6.4% of words have another word with an identical gesture on Qwerty.



(a) Qwerty

(b) GK-D

(c) GK-T

Figure 5.1: Keyboards optimized for gesture typing. The ‘o’ key is shaded to mark the beginning of the word gestures for “or” (white) and “our” (black). (a) The Qwerty keyboard suffers from the problem of gesture ambiguity. Many pairs of words (such as “or” and “our” shown here) share the same gesture on Qwerty. (b) The GK-D keyboard (“Gesture Keyboard—Double optimized”) is the best compromise for gesture clarity and gesture speed. Here, the gestures for “or” and “our” are noticeably different. (c) The GK-T keyboard (“Gesture Keyboard—Triple optimized”) is the best compromise for gesture clarity, gesture speed, and Qwerty similarity. Again, the two gestures are noticeably different.

5.2 Toward Error-Free Gesture Typing

Given Qwerty’s obvious problems, rearranging the keys to make word gestures more distinct should reduce the error rate when gesture typing. However, a layout optimized for gesture clarity (distinctness) might increase the length of each gesture (reducing typing speed) or may be difficult for users to learn. Many questions arise when deciding whether or not to introduce a new layout for gesture typing. For example, if the layout is exclusively optimized for clarity, to what degree will it improve in accuracy over Qwerty? What is the relationship between gesture clarity and gesture typing speed? Can we design an optimized

layout similar to Qwerty in order to ease the learning process?

To answer these questions, we explore the layout optimization space related to gesture typing by applying a rigorous mathematical optimization. Our research not only deepens the understanding of the optimization space of gesture typing, but also contributes a set of optimized layouts that significantly outperform Qwerty (in terms of both gesture clarity and speed) and can immediately benefit mobile device users.

5.3 Optimization Metrics

Gesture Clarity

The gesture clarity metric is the most important metric in our optimization. The purpose of this metric is to measure how unique the word gestures on a keyboard layout are. We based the metric on the location channel in SHARK^2 [Kristensson and Zhai 2004] and represent each word's gesture as its ideal trace, the polyline connecting the key centers of the word's letters. We define the nearest neighbor of a word w to be the word whose ideal trace is closest to w's ideal trace. This is the word that is most likely to be confused with w when gesture typing, independent from the language model. The closer a word is to its nearest neighbor, the more likely its gesture will be not be recognized properly. The gesture clarity metric score for a given keyboard layout is simply the average distance (weighted by words' frequencies) between each word and its nearest neighbor on that keyboard layout:

$$\text{Clarity} = \sum_{w \in L} f_w d_w, \text{ where } d_w = \min_{x \in L - \{w\}} d(w,x) \text{ and } \sum_{w \in L} f_w = 1. \tag{5.1}$$

L is a 40,000-word lexicon, f_w is the frequency of w, and d_w is the distance between w and its nearest neighbor. We compute the distance between two ideal traces w and x via proportional shape matching. Each gesture is sampled into N equidistant points, and the distance is simply the average of the distance between corresponding points:

$$d(w,x) = \frac{1}{N}\sum_{i=1}^{N} \|w_i - x_i\|_2. \tag{5.2}$$

Time Complexity Refinements

Since the gesture clarity metric compares the gestures of every pair of words to find each word's nearest neighbor, its time complexity is $\mathcal{O}(N \cdot |L|^2)$. Here, L is the number of words in the lexicon and N is the number of sample points in each word gesture. Its quadratic time complexity with respect to L stands in stark contrast to the time complexities of earlier optimization metrics which are exclusively linear with respect to L, making optimization using it intractable. For our 40,000-word lexicon, there are nearly 800 million pairs of word gestures to compare for each keyboard layout that we examine during the optimization process.

We made two key algorithmic refinements to make the metric more tractable. First, when searching for the nearest neighbor for each word, we only considered prospective neighbors that started and ended with characters that were located within one key diagonal of the word's starting and ending character, respectively. This is similar to the initial template-pruning step employed in SHARK^2 [Kristensson and Zhai 2004], where the dis-

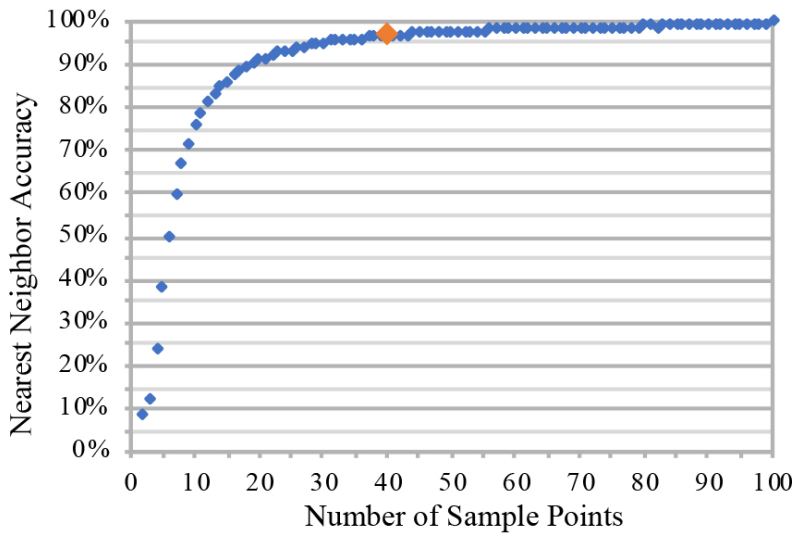


Figure 5.2: Word gesture neighbor sensitivity. The nearest neighbor that we find for a word depends on how finely the word gestures are sampled. Here, we show the percentage of nearest neighbors that are the same as when 100 sample points are used. The red dot signifies 40 points, the amount we used.

tance threshold in this case is the diagonal length of a key. Second, we used a small number of gesture sample points N to represent each word's gesture. If N were too large, the computation would be very expensive. If N were too small, word gestures (especially longer ones) might not be represented properly, leading to incorrectly chosen nearest neighbors.

In order to see how small we could make N without affecting the integrity of our results, we performed a small experiment. First, we found each word's nearest neighbor on Qwerty using very fine sampling (N = 100). Then, we repeated this step for smaller values of N down to N = 20 and counted the number of nearest neighbors that were identical to the N = 100 case. Figure 5.2 shows the results. When the number of sample points is reduced to 40, 96.9% of the nearest neighbors are the same as they were before. We used this value for N in our algorithm.

Gesture Speed

The gesture speed metric estimates how quickly users can gesture type on a keyboard layout. We based this metric on the CLC model by Cao and Zhai [Cao and Zhai 2007]. The model (which stands for "curves, line segments, and corners") stems from human motor control theory, and was designed to predict the amount of time it takes for a person to make an arbitrary pen stroke gesture. To do this, the model partitions the gesture into segments, where each segment is a curve (with a constant radius of curvature), a straight line, or a corner (whose interior angle does not need to be 90°). The time that it takes for a person to gesture each type of segment is modeled with a different function. For line segments, the time is modeled with a power function that echoes how people tend to gesture faster with longer lines:

$$T(\overline{AB}) = m \cdot (\left\|\overline{AB}\right\|_2)^n. \tag{5.3}$$

Here, $\overline{AB}$ is a line segment, the output T is in milliseconds, $\left\|\overline{AB}\right\|_2$ is the length of $\overline{AB}$ in millimeters, and both m and n are constants, which were found to be 68.8 and 0.469 respectively in Cao and Zhai's original formulation.

A polyline gesture is simply a collection of individual line segments. The time to complete this type of gesture is modeled as simply the sum of the individual line segments' functions:

$$T(P) = \sum_{\overline{AB} \in P} T(\overline{AB}), \tag{5.4}$$

where P is the polyline and $\overline{AB}$ is a segment in the polyline. Although Cao and Zhai found that the angles between polyline segments (that is, of a polyline's corners) have an effect on gesture entry time, the magnitude of the effect was small: less than 40 ms per corner compared to 200–700 ms per segment. Hence, the model omits uses corners to delineate segments but omits their 40 ms contribution.

As with the gesture clarity metric, each word in the lexicon is represented as its ideal trace. To help compute the metric, we store a table of the weighted number of occurrences of each bigram in our lexicon. The weighted number of occurrences $o(i\text{—}j)$ of a bigram $i\text{—}j$ (for letters i and j) is calculated as follows:

$$o(i\text{—}j) = \sum_{w \in L} f_w \cdot (\text{\# occurrences of } i\text{—}j \text{ in } w). \tag{5.5}$$

Here, L is the lexicon, w is a word in the lexicon, and f_w is the frequency of word w in L.

Each bigram is represented by a different line segment in the CLC model. Hence, to estimate G, the average time it takes to complete a word gesture, we calculate the following:

$$G = \sum_{i,j \in \alpha} o(i\text{—}j) \cdot T(\overline{K_i K_j}). \tag{5.6}$$

Here, i and j are both letters in alphabet α, the set of lowercase letters from 'a' to 'z.' K_i and K_j are the key centers of the i and j keys, respectively, $\overline{K_i K_j}$ is the line segment connecting the key centers, and the function T is defined in Equation 5.4. Hence, G is measured in milliseconds.

The last step is to convert the gesture duration G into words per minute (WPM), a

measure of typing speed. Doing so gives us our gesture speed metric score:

$$\text{Speed} = \frac{60,000}{G}. \tag{5.7}$$

60,000 represents the number of milliseconds in one minute. When calculating the gesture typing speed of a keyboard layout, we do not consider the effects of the space bar, capitalization, or the Shift key. One of the gesture typing's advantages over touch typing, in fact, is that spaces are automatically added between word gestures, eliminating the need for one in approximately every 5.7 characters typed [Zhai and Kristensson 2003]. Moreover, most of today's gesture-typing systems apply capitalization and diacritics automatically.

We should also note that, because the CLC model omits the cost of gesturing corners and the cost of traveling from the end of one gesture to the beginning of the next, the calculated speeds generally overestimate the speeds at which users would actually type. Rick [Rick 2010] proposed an alternative to the CLC model that is also based on Fitts's law, and although we ultimately chose to use the CLC model for our metric, we implemented Rick's model (without key taps for single-character words) to compare the models' behaviors. We found that Rick's model consistently output lower speed estimates than the CLC model, but that they both followed the same overall trend. More specifically, the mean (std. dev.) ratio between Rick's model's predicted speeds and the CLC model's predicted speeds for our final set of optimized layouts is 0.310 (0.004). After normalizing the metrics as described on the next page, the mean (std. dev.) ratio becomes 0.995 (0.016).

Qwerty Similarity

As has been thoroughly studied [Rick 2010; Yamada 1980; Zhai, Hunter, and Smith 2002], the key obstacle to the widespread adoption of optimized layouts is the arduous process of learning the new layouts. The Qwerty similarity metric measures how similar a given keyboard layout is to Qwerty. By making a new layout more similar to Qwerty — and hence less alien to longtime users of Qwerty — we hope to bridge the gap between the short-term frustration of learning the new layout and the long-term benefits that the layout provides.

The metric is based on the constraint that Bi, Smith, and Zhai [Bi, Smith, and Zhai 2010] used when creating the Quasi-Qwerty layout. In that optimization, which was for typing speed only, keys were not allowed to move more than one slot away from their Qwerty locations. Dunlop and Levine [Dunlop and Levine 2012] later relaxed this constraint in their multi-objective keyboard optimization by using the total squared Euclidean distance between keys' positions and their Qwerty locations instead. Since a keyboard layout is essentially a grid of keys, we use the total Manhattan distance between keys' positions and their Qwerty locations to measure Qwerty similarity. Like Dunlop and Levine's metric, this allows more freedom than the hard constraint used by Quasi-Qwerty. However, unlike Dunlop and Levine's metric, individual keys are not punished so severely if they move far from their Qwerty locations. This allows us to consider layouts in which a few keys move very far from their Qwerty locations.

The Qwerty similarity metric for a given keyboard layout is computed as follows:

$$\text{Qwerty Similarity} = \sum_{i \in \alpha} (|k_{i_x} - q_{i_x}| + |k_{i_y} - q_{i_y}|), \tag{5.8}$$

where i is a letter in alphabet α, the set of lowercase letters from 'a' to 'z,' and k_{i_x} and q_{i_x} are the x-indices of the i key on the given keyboard layout and Qwerty, respectively. Unlike K_i and K_j in Equation 5.6, which are points whose coordinates are specified in millimeters, k_i and q_i are unit-less ordered pairs of integers that represent the 2D index of key i's slot in the keyboard grid. In most of today's touchscreen keyboard layouts, the second and third rows are offset from the first row by half of a key width. Hence, in order to properly calculate the Manhattan distance for this metric, we treat the second and third rows as if they are shifted to the left by another half of a key width so that the second row is left-aligned with the first row. The resulting representation of keyboard layouts is identical to the one used for creating Quasi-Qwerty [Bi, Smith, and Zhai 2010]. The Qwerty similarity metric is the only one that uses this modified keyboard representation.

5.4 Optimization Procedure

We frame the problem of designing a touchscreen keyboard for gesture typing as a multi-objective optimization in which the three objectives are improving (1) gesture clarity, (2) gesture speed, and (3) Qwerty similarity. There are multiple ways of judging how well a layout meets these objectives. One way is to create a simple objective function that somehow combines the objectives' associated metric scores; for example, by summing the scores in a linear combination. Such an approach, however, would force us to decide how

much each metric should count for in deriving a single optimal layout, when in fact we are more interested in understanding the behavior of each of the metrics and the inherent tradeoffs between them.

As a result, although we still employ a simple objective function as part of our optimization's second phase, we use another approach called Pareto optimization for the optimization at large. Pareto optimization has recently been used to optimize both keyboard layouts [Bi, Smith, and Zhai 2012] and keyboard algorithms [Bi, Ouyang, and Zhai 2014]. In this approach, we calculate an optimal set of layouts called a Pareto optimal set or a Pareto front. Each layout in the set is Pareto optimal, which means that none of its metric scores can be improved without hurting the other scores. If a layout is not Pareto optimal, then it is dominated, which means that there exists a Pareto optimal layout that is better than it with respect to at least one metric and no worse than it with respect to the others. By calculating the Pareto optimal set of keyboard layouts rather than a single keyboard layout, we can analyze the tradeoffs inherent in choosing a keyboard layout and give researchers the freedom to choose one that best meets their constraints.

Our optimization procedure is composed of three phases, described in detail in the subsections below.

Phase 1: Metric Normalization

In the first phase, we perform a series of optimizations for each metric individually to estimate the minimum and maximum possible raw values for each metric. We then normalize each of the metric's scores in a linear fashion so that the worst possible score is mapped

to 0.0 and the best possible score is mapped to 1.0. Normalizing the scores allows us to weight the metrics appropriately in Phase 2.

We use local neighborhood search to perform the optimizations. In order to more reliably find the global extrema instead of local extrema, we incorporate a simulated annealing process similar to the Metropolis random walk algorithm [Hastings 1970; Zhai, Hunter, and Smith 2000]). Each optimization starts with a random keyboard layout using the same footprint as Qwerty and runs for 2,000 iterations. At each iteration, we swap the locations of two randomly chosen keys in the current layout to create a new candidate layout. If the new layout is better than the current layout, we keep the new layout with 100% probability. Otherwise, we only keep the new layout with a probability specified by a user-controlled "temperature." Higher temperatures increase this probability and allow us to escape from local extrema.

In total, we performed 10–30 optimizations for each metric. We found that the range for the raw gesture typing clarity metric scores was [0.256 key widths, 0.533 key widths], that the range for the raw gesture typing speed metric scores was [50.601 WPM, 77.929 WPM], and that the range for the raw Qwerty similarity metric scores was [0, 148]. Qwerty's raw scores for the three metrics are 2.390 mm, 62.652 WPM, and 0, respectively.

Phase 2: Pareto Front Initialization

In this phase, we generate an initial Pareto front of keyboard layouts by performing even more local neighborhood searches. The searches are identical to the ones we perform in Phase 1, except this time we seek to maximize the score from linear combinations of

all three metric scores. We use twenty-two different weightings for the linear combinations and perform roughly fifteen full 2,000-iteration local neighborhood searches for each weighting. The purpose is to ensure that the Pareto front includes a broad range of Pareto optimal keyboard layouts.

The Pareto front starts out empty at the very beginning of this phase, but we update it with each new candidate keyboard layout that we encounter during the searches (at each iteration of each search). To update the front, we compare the candidate layout with the layouts already on the front. Then, we add the candidate layout to the front if it is Pareto optimal, potentially displacing layouts already on the front that are now dominated by the candidate layout. The candidate layout is added whether it is ultimately kept in the particular local neighborhood search or not. This approach is similar to the one that Bi et al. [Bi, Ouyang, and Zhai 2014] used to optimize keyboard correction and completion algorithms.

Phase 3: Pareto Front Expansion

In the last phase, we perform roughly 200 passes over the Pareto front to help "fill out" the front by finding Pareto optimal layouts that are similar to those already on the front. In each pass, we swap two keys in each layout on the front to generate a set of candidate layouts, then update the front with any candidate layouts that are Pareto optimal. This phase is similar to the optimization used by Dunlop and Levine [Dunlop and Levine 2012]. However, by including Phase 2, we can ensure that all possible solutions are reachable without the need to swap more than two keys at a time.

Optimization Parameters

We based our optimization's keyboard representation on dimensions of the Nexus 5 [LG USA 2018] Android keyboard. Since most of today's touchscreen keyboards have very similar profiles, our results should be applicable to any touchscreen keyboard. Each key is represented by its entire touch-sensitive area, with boundaries placed between the center points of neighboring keys, and is 109×165 px (6.22×9.42 mm) in size.

Our lexicon consists of 40,000 words. Before starting the optimization, we converted words with diacritics to their Anglicized forms ("naïve" to "naive," for example), removed all punctuation marks from words (such as "can't"), and made all words completely lowercase. Since gesture typing systems automatically handle diacritics, capitalization, and punctuation marks within words, this should not hurt the integrity of our optimization.

Optimization Runtime

Due to the complexity and scope of our work, it took four machines (with 32 threads apiece) running continuously over the course of nearly three weeks to obtain the results presented below.

5.5 Optimized Keyboard Layouts

Figure 5.3 shows the final Pareto front of keyboard layouts optimized for gesture typing. Overall, the front is composed of 1,725 keyboard layouts chosen from the 900,000+ candidate layouts that we examined in all. No single layout on the front is better than all of the others — each layout is better than the others in some way, and the tradeoffs that are

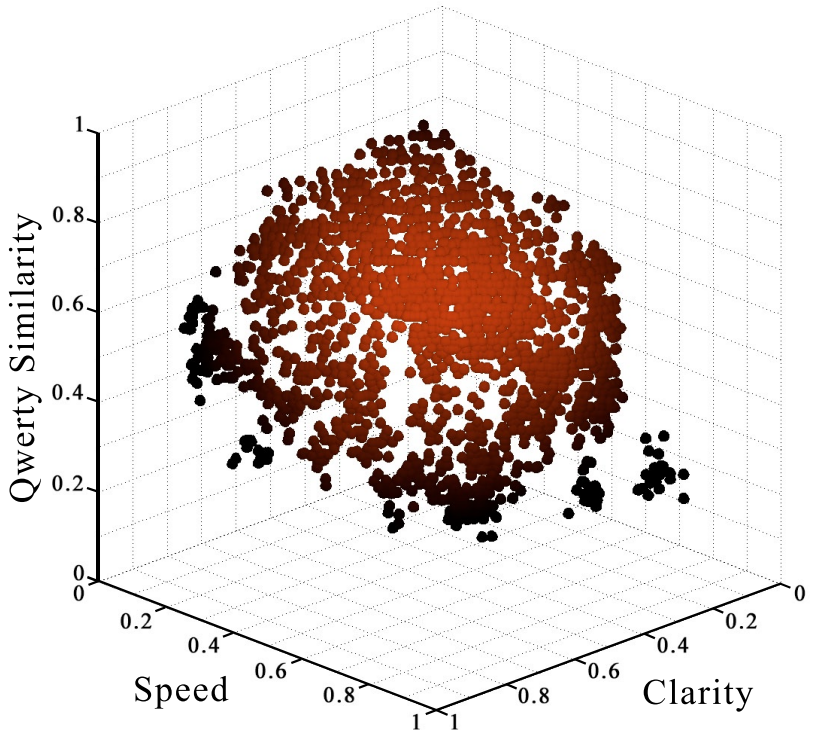
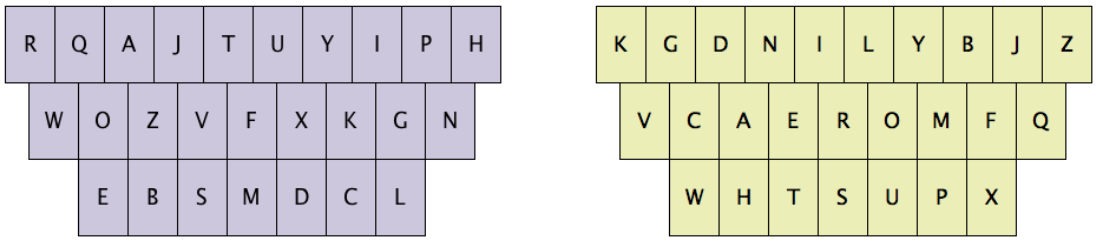


Figure 5.3: 3D Pareto front of keyboard layouts optimized for gesture typing. The keyboard layouts with lighter colors are farther from the origin.

inherent in choosing a suitable layout from the front are reflected in the front's convex shape.

More specifically, the front can be viewed as a three-dimensional design space of performance goals that one can choose from for different usage scenarios. Layouts with high gesture clarity scores, gesture speed scores, and Qwerty similarity scores are more apt to exhibit lower error rates, expert-level gesture entry times, and initial gesture entry times (respectively) than those with low scores. However, since each layout on the front represents a compromise between these three goals, the choice of layout for a particular user or usage scenario depends on the relative importance of each goal. For example, a fast but less accurate user may prefer a layout biased towards clarity, while a user who gesture types very accurately may prefer a layout biased toward speed. Nevertheless, if we know



(a) GK-C (b) GK-S

Figure 5.4: Single-optimized keyboard layouts. (a) Our GK-C keyboard ("Gesture Keyboard—Clarity") is optimized for gesture typing clarity only. (b) Our GK-S keyboard ("Gesture Keyboard—Speed") is optimized for gesture typing speed only.

nothing about users' preferences or wish to choose a layout that can best accommodate a wide variety of preferences, it is reasonable to use one that is in the middle of the convex surface (serving each goal on a roughly equal basis) as Dunlop and Levine did [Dunlop and Levine 2012].

We will now highlight layouts optimized for each of the three metrics as well as layouts that serve roughly equal combinations of metrics. These layouts may serve as useful references to researchers and designers and will help us test the effectiveness of our optimization and its associated metrics in the user study.

Single-Optimized Keyboard Layouts

Figure 5.4(a) shows GK-C ("Gesture Keyboard—Clarity"), the layout optimized exclusively for gesture typing clarity. Figure 4(b) shows GK-S, which was optimized exclusively for speed. The layout optimized for Qwerty similarity is simply Qwerty itself, and is shown in Figure 5.1(a).

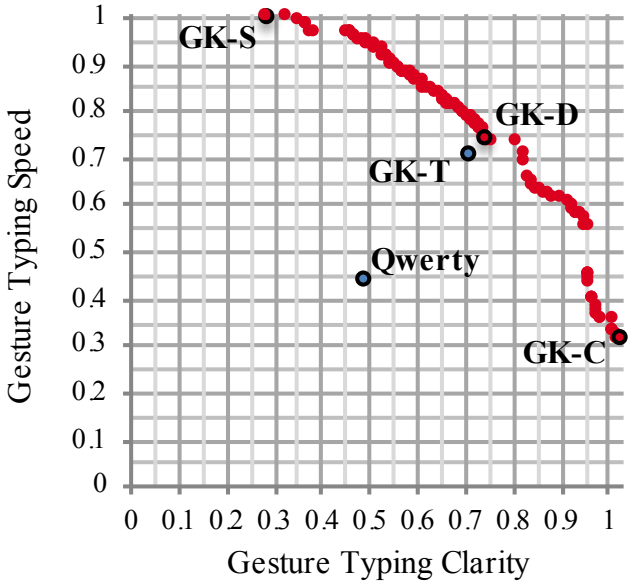


Figure 5.5: 2D Pareto front for gesture typing clarity and gesture typing speed. GK-D, our double-optimized layout, is the point on the front nearest the 45° line. Note that Qwerty is far worse in both dimensions than GK-D, and that GK-T (which accommodates yet another dimension) is only slightly worse on these two dimensions than GK-D.

Double-Optimized Keyboard Layout

Figure 5.1(b) shows GK-D (where the 'D' stands for "double-optimized"). This keyboard offers a roughly equal compromise between gesture typing clarity and gesture typing speed without regard to learnability (Qwerty similarity). To find this layout, we projected the three-dimensional Pareto front onto the clarity–speed plane to derive a 2D Pareto front between clarity and speed, then chose the layout on the 2D front that was closest to the 45° line. Figure 5.5 shows the 2D Pareto front and GK-D.

Triple-Optimized Keyboard Layout

Figure 5.1(c) shows GK-T, where the 'T' stands for "triple optimized." This keyboard offers a roughly equal compromise between all three metrics: gesture typing clarity, gesture

typing speed, and Qwerty similarity. It is the one on the three-dimensional Pareto front that is closest to the 45° line through the space. As Figure 5.5 illustrates, it is possible to accommodate the extra dimension of Qwerty similarity without a big sacrifice to clarity and speed.

Discussion

Table 5.1 shows the metric scores for our optimized layouts as well as previous optimized layouts. Together, these optimized layouts give us a good understanding of what is possible in the optimization space for gesture typing.

First, we can improve gesture clarity by 38.8% by optimizing for clarity alone: GK-C's raw metric score is 0.543 key widths while Qwerty's is 0.391 key widths. Likewise, we also see that we can improve gesture speed by 24.4% by optimizing for speed alone (resulting in GK-S).

Second, the 2D Pareto front for gesture clarity and gesture speed (Figure 5.5) shows that these two metrics conflict with each other. It forms a roughly -45° line, indicating that optimizing for one leads to the decrease in the other. As GK-C and GK-S illustrate, the clarity metric tends to arrange common letters far apart in a radial fashion while the speed metric clusters common letters close together.

However, despite the conflict, it is possible to arrange common letters close together while keeping word gestures relatively distinct, achieving large improvements in both clarity and speed. In GK-D (our double-optimized keyboard), letters in common n-grams such as "the," "and," and "ing" are arranged together while the n-grams themselves are spaced

Layout	Gesture Typing Clarity		Gesture Typing Speed		Qwerty Similarity	
	Normalized	Raw (key widths)	Normalized	Raw (WPM)	Normalized	Raw (slots)
Qwerty	0.489	0.391	0.441	62.653	1.000	0
Sath Trapezoidal [Dunlop and Levine 2012]	0.568	0.413	0.704	69.843	0.730	40
GK-C	1.038	0.543	0.317	59.256	0.554	66
GK-S	0.283	0.334	1.000	77.940	0.311	102
GK-D	0.743	0.462	0.739	70.793	0.324	100
GK-T	0.709	0.452	0.704	69.830	0.716	42
Square ATOMIK [Zhai, Hunter, and Smith 2002]	0.362	0.356	0.878	74.591	N/A	N/A
Square OSK [Rick 2010]	0.381	0.361	0.979	77.358	N/A	N/A

Table 5.1: Keyboard metric score comparison. Shaded rows signify previous layouts.

apart. This arrangement offers a 17.9% improvement in gesture clarity and a 13.0% improvement in gesture speed over Qwerty.

Third, accommodating Qwerty similarity (as GK-T does) does little harm to gesture clarity or gesture speed. GK-T's gesture clarity is only 0.01 key widths lower than GK-D's, and GK-T's predicted speed is only 1 WPM lower than GK-D's. Meanwhile, GK-T's Manhattan distance from Qwerty is just 42 key slots, while GK-D's is 102 key slots.

Comparison with Previous Optimized Layouts

The key difference between our proposed keyboard layouts and previous optimized layouts is that our layouts are optimized for multiple gesture typing factors while previous layouts are predominantly optimized for tapping speed. As Table 1 shows, previous layouts such as Sath Trapezoidal [Dunlop and Levine 2012], Square ATOMIK [Zhai, Hunter, and Smith 2002], and Square OSK [Rick 2010] have high gesture speed scores but low gesture clarity scores.

5.6 User Study

Since the main focus of this work is to computationally discover the optimization space for gesture typing, the conclusions that we have made so far are based on theoretical metrics. Of the three metrics that we established, only one — gesture speed — is based on a model that directly predicts its respective performance goal, which in its case is words per minute. The others, gesture clarity and Qwerty similarity, do not directly measure their performance goals: error rate and learnability, respectively. Hence, we performed an empirical study to

give us a sense of how the metric scores correspond to real performance and whether the optimization itself is effective.

Experimental Setup

In the study, participants gesture typed a set of 22 words with each keyboard layout using a Nexus 5 [LG USA 2018] smartphone in portrait mode. As in Bi et al. [Bi, Smith, and Zhai 2012], participants had to gesture each word seven times in succession. We instructed the participants to gesture as quickly as possible and ignore any errors, which, for us, achieved two goals. First, it allowed us to stress test the keyboard's gesture decoder by providing it very sloppy gestures; the resulting data is also more differentiable in evaluating accuracy. Second, it simulated a type of expert input behavior: entering words first and coming back to fix mistakes later.

Our study was a within-subject design that tested three keyboard layouts: Qwerty (our baseline), GK-D (the roughly equal compromise for clarity and speed only), and GK-T (the roughly equal compromise for clarity, speed, and Qwerty similarity). To conduct the experiment, we created Android implementations of GK-D and GK-T based on the Android [Android Open Source Project 2018] keyboard, and developed an Android application (Figure 5.6) to collect users' gesture typing data.

All participants started with Qwerty but used the other two layouts in alternating order. The first three words served as a warm-up phase to familiarize participants with the task (we did not collect their data), and the other 19 words are from the list proposed by Zhai and Kristensson [Zhai and Kristensson 2008]: "*the and you that is in of know not they*

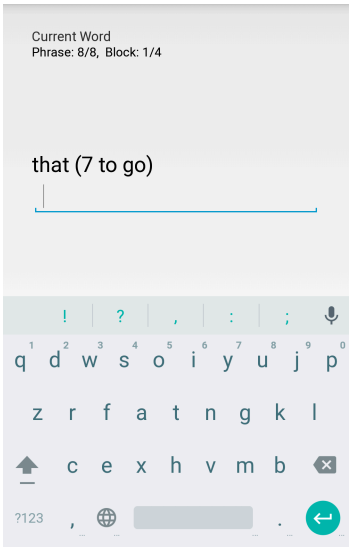


Figure 5.6: Gesture typing user study application. The layout shown is GK-T.

get have were are bit quick fox jumps lazy." These words cover all letters of the English alphabet and approximate both letter frequencies and digraph frequencies in English. They were divided into groups of four or five with short breaks in between.

To measure gesture typing accuracy, we compare each committed word with the respective requested word using a strict binary string equality comparison. The committed word is the word that appears after the participant lifts his or her finger from the screen and the keyboard algorithm applies any word corrections that it sees fit. To measure gesture entry times, we recorded either (1) the length of time from when a word was presented on the screen to when the word was committed (for the first repetition of a word), or (2) the length of time between when the last word was committed to when the current word is committed (for subsequent repetitions of a word).

The entry times for the first repetitions of words offer a rough (but by no means perfect) perspective of our keyboards' learnability, while the entry times for latter repetitions of the word are a rough estimate of expert-level entry times. The rationale for the latter is that by

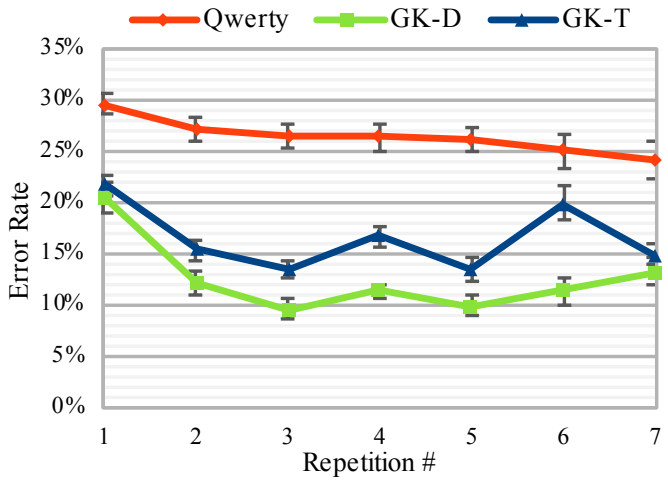


Figure 5.7: Error rates across 14 participants for Qwerty, GK-D, and GK-T. GK-D's and GK-T's average error rate is 52% and 31% less than Qwerty's, respectively. Error bars indicate standard errors.

repeating the same word in a row, users will reach a stage where the input behavior is mostly governed by motor control ability, which reflects expert input behavior. However, this is only a limited proxy for the study of the complex learning process and expert-level typing performance at scale, which may require a longitudinal logging study of real keyboard use, notwithstanding privacy and other challenges associated with such methods.

A total of 14 volunteers (9 female, 5 male) participated in the experiment. Three were age 18–25, nine were 26–35, and two were 36–45. Eight of them primarily use Android smartphones and the rest iPhones. Thirteen were at least somewhat familiar with gesture typing, and five were at least somewhat familiar with alternative keyboard layouts. All of them were right-handed. Each experiment lasted less than an hour.

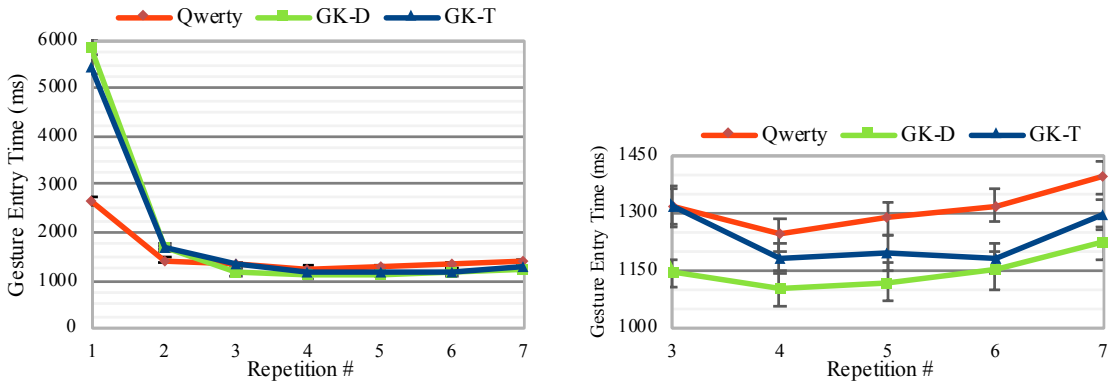
Experimental Results

Error Rate

Figure 5.7 shows participants' overall error rates with each layout and how those error rates changed as participants made successive repetitions of each word. The mean (std. dev.) error rate for Qwerty, GK-D, and GK-T were 26.4% (7.2%), 12.6% (6.6%), and 16.6% (6.2%), respectively. This means that the error rates for GK-D and GK-T were 52% and 37% less than Qwerty, respectively. The keyboard layout has a significant main effect on the overall error rate ($F_{2,26} = 35.46$, $p < 0.001$). Pairwise mean comparison over all repetitions showed that the differences were significant ($p < 0.01$) for every pair of keyboards except GK-D vs. GK-T. For Repetitions 2–6, however, the difference is significant ($p < 0.01$) for every pair of keyboards.

Initial Gesture Entry Time

Figure 5.8 shows how long, on average, it took participants to gesture words per repetition $\times$ layout. We noticed that participants often planned out their gestures in the first repetition, but resorted to motor memory in later repetitions. The mean (std. dev.) initial entry time was 2,655 ms (502 ms), 5,870 ms (1,190 ms), and 5,468 ms (1,140 ms) for Qwerty, GK-D, and GK-T, respectively. The keyboard layout has a significant main effect on the initial entry time ($F_{2,26} = 75.26$, $p < 0.001$). Pairwise mean comparison showed that the differences were significant ($p < 0.01$) for each pair of keyboards except GK-D vs. GK-T.



(a) All repetitions

(b) Repetitions 3–7 only

Figure 5.8: Gesture entry times across 14 participants for Qwerty, GK-D, and GK-T. Error bars indicate standard errors. (a) The initial entry time (Repetition 1) using GK-D and GK-T is over twice as long as it is using Qwerty. (b) For Repetitions 3–7 (approximating expert usage), the average entry time for GK-D and GK-T are 12.5% and 6.0% faster respectively than they are for Qwerty.

Expert-Level Gesture Entry Time

Figure 5.8(b) shows the expert-level entry time (Repetitions 3–7) in detail. The mean (std. dev.) entry time in this case is 1,315 ms (300 ms) for Qwerty, 1,150 ms (333 ms) for GK-D, and 1,237 ms (310 ms) for GK-T. The keyboard layout has a significant main effect on the expert-level entry time ($F_{2,26} = 12.46$, $p < 0.001$). The expert-level entry time for GK-D and GK-T is 12.5% and 6.0% faster than that for Qwerty. Pairwise mean comparison showed the differences were significant ($p < 0.01$) for each pair of keyboards except GK-D vs. GK-T.

Discussion

The results from the user study lead to several findings, although we again stress that they are limited by the fact that our experiment was conducted in a single session. First, keyboards optimized for gesture clarity are more accurate than those without. The error rates

for GK-D and GK-T are 52% and 37% less than Qwerty, respectively.

Second, including both gesture typing clarity and gesture typing speed in the optimization process results in layouts that outperform Qwerty in terms of both accuracy and expert typing speed. Both GK-D and GK-T significantly outperform Qwerty in both of these metrics. Third, considering the Qwerty similarity metric has only minor effects on accuracy and speed. The differences we observed between the expert-level entry times for GK-D and GK-T were not statistically significant. Finally, the Qwerty similarity metric is not very effective in improving learnability. Though the mean initial entry time for GK-T was lower than that of GK-D, we did not observe a statistically significant difference between the two. This is likely due to the relative leniency of the Qwerty similarity metric compared to Quasi-Qwerty's hard constraint [Bi, Smith, and Zhai 2010] and Dunlop and Levine's squared distance metric [Dunlop and Levine 2012].

The findings also give us a better sense of how a layout's gesture clarity and Qwerty similarity scores correspond to real performance. Recall that the gesture speed scores are based on an empirically derived model. For example, the gesture clarity score increase from 0.489 in Qwerty to 0.743 in GK-D (an increase by 0.254 — see Table 5.1) corresponds to a decrease in the mean error rate from 26.4% to 12.6%. Yet, the Qwerty similarity score increase from 0.324 to 0.716 does not improve learnability as we have defined it, while the increase from 0.716 to 1.000 drastically does.

Still, we do not know the exact relationship between these two metrics' scores and the corresponding real-world performance measures. Conversely, the problem of finding metrics that empirically model gesture typing error rate and keyboard learnability remains to be solved. In the case of learnability, Quasi-Qwerty's constraint improves it [Bi, Smith,

and Zhai 2010] but cannot be used as a continuous model, and squared distance has not yet been shown to model it in its various degrees. Determining those relationships requires empirically testing many more layouts using a variety of values for each metric score, and remains promising future work.

5.7 Discussion

The present work, for the first time, defines a multidimensional optimization space for gesture typing — comprising gesture clarity, gesture speed, and Qwerty similarity — and systematically explores that space. In the process, we contribute a set of optimized layouts such as GK-D (optimized for both gesture clarity and gesture speed) and GK-T (optimized for gesture clarity, gesture speed, and Qwerty similarity) that can immediately benefit users. Though limited, our empirical study of these layouts led to the following findings.

First, optimizing the layouts for gesture clarity drastically improves gesture typing accuracy. By incorporating gesture clarity as an optimization dimension, GK-D and GK-T reduced error rates by 52% and 37% over Qwerty, respectively. Second, gesture clarity and gesture speed conflict with each other, but despite the conflict, incorporating both in the optimization process leads to superior performance over Qwerty with respect to both metrics. GK-D and GK-T, for example, improved expert-level entry times by 12.5% and 6.0% over Qwerty, respectively. Third, Qwerty similarity as we have defined it has only a minor conflict with gesture clarity and gesture speed, but is not effective in improving learnability.

We discuss future areas of gesture typing research in Section 7.2.

Chapter 6

The RAD: Making Racing Games Equivalently Accessible to People Who Are Blind

As we have mentioned in the Introduction and in the previous chapter, personalization is one of the keys to reducing input overhead. This is especially true in the case of accessibility, since by definition we must personalize computers for each user's particular circumstance to make them accessible. For users with disabilities, personalization can also significantly reduce output overhead. In this section, we develop a user interface that helps make our world — and in particular our intangible virtual world — accessible to people who are blind, allowing them to control a real-time system with an efficiency very similar to that of people who are sighted.

This contribution differs from this dissertation's previous contributions in an important way. While the previous contributions were dedicated to making the process of providing *inputs* to computers more seamless and invisible, this contribution focuses on achieving parity with respect to a computer's *outputs*; namely, the display (or feedback) that a computer responds with. Doing so is challenging because computers must sacrifice much of their display expressiveness or "throughput" in order to accommodate disabilities such as blindness.

This chapter's focus will be on a challenging domain: making racing games equally accessible to people who are blind. Exploring video games allows us to address an impor-

tant instance of the throughput expressiveness parity problem. Since video games are very much visual experiences, we must somehow give players who are blind an equivalent experience, including the ability to make the same moment-to-moment decisions as sighted players make. Moreover, we must do so using a much less expressive mode of output or *channel*: in our case, audio. The reason that we focus on racing games in particular is that they are very time-sensitive, requiring players to make racing decisions in real time.

The Need for Equivalently Accessible Games

Accessibility alone is not enough to make the world a fair place for people with disabilities. Even with assistive technologies, if people with disabilities cannot experience the world in the same manner as anyone else [Steinfeld and Maisel 2012; WBDG Accessible Committee, Maisel, and Ranahan 2017], or even as productively as anyone else [Hedgpeth, Black Jr., and Panchanathan 2006], the world will not yet be fair — or as we will say, *equivalently accessible*.

Imagine a wheelchair ramp leading to an entrance of a public library. Technically, the ramp would make the library accessible to people using wheelchairs. But if that ramp makes such a circuitous route on its way up that only people who need it would ever want to use it, the ramp would not make accessing the library efficient or fair [WBDG Accessible Committee, Maisel, and Ranahan 2017]. Worse, suppose that the ramp leads to a separate, less handsome entrance to the library or, even worse, to a different building altogether: a smaller, adjacent library that only has the digest versions of books from the main library. These facilities would clearly not be fair for people using wheelchairs.

Figure 6.1: Study participant P8 — who is congenitally blind — playing our racing game prototype using the racing auditory display (RAD). The RAD outputs spatialized sound and works with a standard pair of headphones. Using the RAD, players who are blind can play the same types of racing games that sighted players can play with an efficiency and sense of control that are similar to what sighted players have. Our supplemental video shows P8 using the RAD with the RAD's audio included.

This situation, however, is similar to what video games are like for people who are blind. Most blind-accessible games today are either loaded with competing sources of information that players must sift through [Atkinson and Gucukoglu 2004; GMA Games 2005; Shultz 2015; Westin 2004], slowing down the efficiency of play, or are very simplified versions of games that sighted players would play [Allman, Dhillon, Landau, and Kurniawan 2009; audiogames archive 2015; Kim and Ricaurte 2011; Miller, Parecki, and Douglas 2007; Morelli, Foley, and Folmer 2010; Shultz 2014a; Yuan and Folmer 2008], to the extent that the player may be doing nothing more than following orders from the game [Allman, Dhillon, Landau, and Kurniawan 2009; Kim and Ricaurte 2011; Miller, Parecki, and Douglas 2007; Shultz 2014a; Yuan and Folmer 2008]. These games are technically accessible to players who are blind, but they are far from the same game that sighted

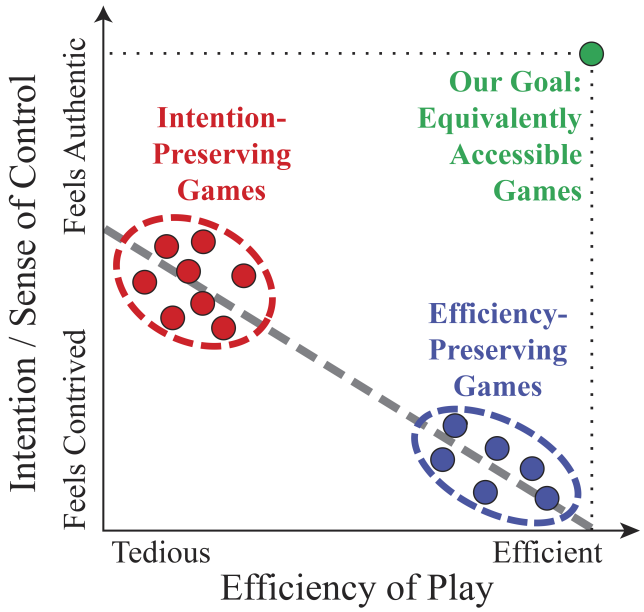


Figure 6.2: The intention–efficiency tradeoff. When designing blind-accessible games, game designers must choose between sacrificing each game's complexity — and by extension the player's *intention* or sense of control within the game — and the game's efficiency of play. Moreover, sophisticated actions such as cutting corners in racing games are difficult to incorporate even in intention-preserving games, so many do not feel fully authentic to play compared to what sighted players would play. Our goal is to overcome this tradeoff to help racing games become equivalently accessible to people who are blind.

players would play, and so are not equivalently accessible.

The Intention–Efficiency Tradeoff

The reason that blind-accessible games struggle to deliver the same experiences as games for sighted players is that there is a fundamental conflict between preserving the game's complexity and preserving the game's pace when designing a blind-accessible version of a game. Preserving the former allows players to have the same sense of control that sighted players have when playing existing games, while preserving the latter keeps the action continuous and in real-time.

Figure 6.2 illustrates this tradeoff, with the sense of control that the game affords to

the player on the vertical axis and the game's efficiency (pace) on the horizontal axis. For a game to be equivalently accessible to people who are blind, it should offer both, as the green dot at the top-right of the figure indicates. In practice, however, designers must sacrifice one of the two, causing existing blind-accessible games to fall into two distinct groups: what we call *efficiency-preserving games* and *intention-preserving games*.

Efficiency-preserving games, indicated by the blue dashed circle in Figure 6.2, are ones that sacrifice the sense of control that they afford players to keep their gameplay moving at a continuous pace. These include games such as *Blind Hero* [Yuan and Folmer 2008], *Rock Vibe* [Allman, Dhillon, Landau, and Kurniawan 2009], and *Blindfold Racer* [Shultz 2014a]. They are often simplified versions of games that sighted players would play and often boil down the gameplay to a simple test of reaction speed. In *Blind Hero* and *Rock Vibe*, for example, players do not get to prepare for upcoming beats like sighted players would when playing *Guitar Hero* or *Rock Band*, which these games were based on. Rather, players are tasked with pressing buttons as soon as they feel corresponding vibration cues.

Intention-preserving games, indicated by the red dashed circle in Figure 6.2, are ones that sacrifice their efficiency of play to maintain more of their complexity and, by extension, give players a greater sense of control. These include *Terraformers* [Westin 2004] and *Blindfold Color Crush* [Shultz 2015]. They are often cumbersome to play because they force players to navigate menus and process many audio cues just to understand what the current situation in the game is at any given time. Moreover, although they preserve much of their complexity and sense of control, they cannot preserve it all: complex actions such as cutting corners and performing head shots remain out of reach.

Chapter Contributions

In this chapter, we present the *racing auditory display (RAD)*, an audio-based user interface with the goal of overcoming the intention–efficiency tradeoff to help racing games become equivalently accessible to people who are blind. The RAD comprises two novel sonification techniques: the *sound slider* for understanding a car's speed and trajectory on a racetrack and the *turn indicator system* for alerting players of the direction, sharpness, length, and timing of upcoming turns. Figure 6.1 shows a participant who is congenitally blind playing a racing game with the RAD.

We conducted two user studies to investigate whether the RAD allows players who are blind to play racing games at the same pace and with the same level of control as sighted players can. In the first study, we found that players preferred to play a racing game using the RAD over that of *Mach 1* [audiogames archive 2015], a popular blind-accessible racing game. In the second study, we found that the RAD makes it possible for a gamer who is blind to race as well on a complex racetrack as casual sighted players do. When that gamer raced using the RAD, there was no significant difference between his lap times or driving paths compared to those of casual players racing with sight.

6.1 Intention And Its Role in Racing Games

Here, we introduce the concept of intention to describe what we mean by sense of control more precisely, and will illustrate how this concept applies to racing games. This concept can be used to examine whether a game gives players a high sense of control and, if not, how it can be changed to do so.

Intention is the process of "allowing and encouraging players to do things [within games] intentionally" [Church 1999a; Church 1999b; Church 2006]. More specifically, it is the process of "making an implementable plan of one's own creation in response to the current situation in the game world and one's understanding of the game play options" [Church 1999a; Church 1999b; Church 2006]. By breaking this definition down into parts, we can see that for a game to support intention, it must help the player perform the following three activities:

1. Understand the current situation in the game.
2. Understand what game play options are currently available.
3. Make an implementable plan of their own creation.

These activities are analogous to the three components of Yuan et al.'s game interaction model [Yuan, Folmer, and Harris 2011]. When we say that a game affords players a high sense of control, we mean that the game supports intention, which more precisely means that the game supports the player in performing each of the three activities listed above. For a blind-accessible video game to be *equivalently accessible* to people who are blind, it must support these three elements of intention without sacrificing the game's pace — overcoming the tradeoff in Figure 6.2 — and without simplifying the gameplay.

To support the first activity, racing games must help players understand all aspects of their current situation that are relevant to racing: their vehicle's position and orientation on the racetrack, a general sense of its current speed, the nature of any upcoming turns, etc. The game does not need to help players understand aspects of the current situation that are not relevant to racing, such as their vehicle's paint color or even its precise speed

in mph/kph. (In fact, many games such as the *Grand Theft Auto* series and most of the *Mario Kart* series do not show players their vehicle's speed.) To support the third activity, racing games should make it possible for players to form strategies such as cutting corners or positioning themselves to better handle an upcoming turn.

6.2 The Racing Auditory Display (RAD)

In this section we introduce the *racing auditory display (RAD)*, a user interface whose goal is to help racing games become equivalently accessible to people who are blind. The RAD was designed with the principle that it should not just tell players what to do but rather give them enough relevant information to form a plan of action themselves.

The RAD comprises two novel sonification techniques: the *sound slider* and the *turn indicator system*. The *sound slider* helps the player understand their car's speed and trajectory on a racetrack while the *turn indicator system* alerts players of the direction, sharpness, length, and timing of upcoming turns well in advance of the actual turns. Together, the techniques allow players to understand aspects about the race and perform a wide variety of actions that are not possible to understand and perform in current blind-accessible racing games.

The RAD's Sound Slider

The RAD's sound slider is a novel mechanism for displaying a value within a range using spatialized (3D) sound. It is analogous to a traditional user interface slider, where the slider's track is a line segment in the 3D soundscape and the slider's handle is replaced

with a virtual speaker (sound emitter). The position of the speaker on the virtual track represents the slider's displayed value, where one end of the track represents the slider's minimum value, the other its maximum value, and positions in between intermediate values. The slider's value is for display only: the speaker cannot be manually manipulated by the user like a traditional user interface slider's handle can.

Figure 6.3 shows the specific sound slider configuration that we propose for blind-accessible racing games. The slider's track is a virtual horizontal bar of width w placed a distance d in front of the player's face in the soundscape. In our prototype racing game, w is 65 m and d is 12 m. The speaker emits the sound of the player's car's engine and slides left and right along the bar as the sound slider updates its value.

We explained the concept of the sound slider to our studies' participants as follows. We asked them to imagine being behind the car that they were controlling, so they could hear the sound of the car's engine right in front of their face. The car's sound will move left or right as the car becomes more at risk of hitting the track's left or right edges, respectively. When they steer, they control the car's sound directly, so if they hear the car's sound move far toward the left, they will want to steer right to bring the sound back toward the center.

If the player is in a turn and not turning nearly as sharply as they need to, perhaps because they are going too fast, the sound slider will emit a tire screeching sound adapted from [audible-edge 2009] from the same position as the car's engine sound. This acts to warn the player that they must slow down by hitting the brake or letting go of the accelerator to properly complete the turn.

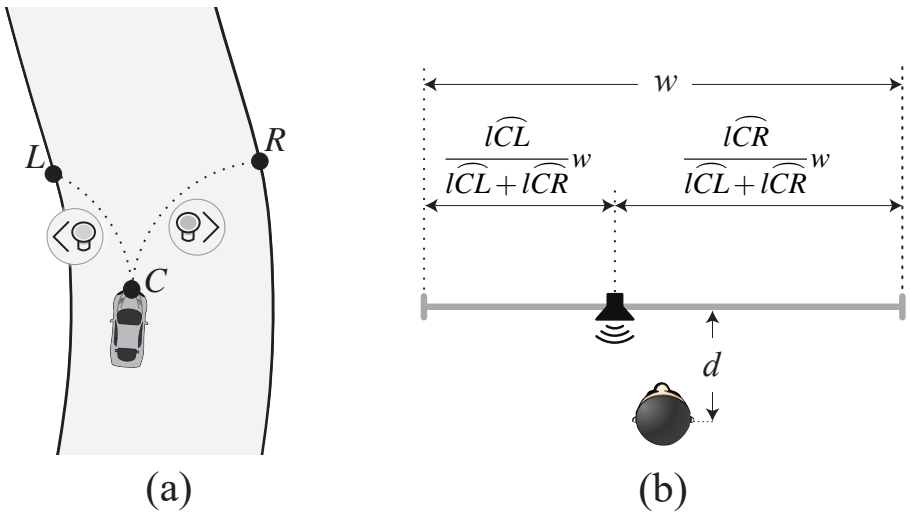


Figure 6.3: The RAD's sound slider. (a) Sample car pose showing what the car's trajectories would be if the player were to steer fully left or fully right. (b) Overhead view of corresponding rendered spatialized (3D) soundscape. The RAD's sound slider is a speaker emitting the car's engine noise whose lateral position in the soundscape tracks the ratio of the trajectories' lengths. The ratio represents the player's relative risk of hitting either edge of the track. In this case, the player will hear the car's engine right in front of their face but slightly to the left.

Computing the Slider Value

Figure 6.3 illustrates how the RAD computes the sound slider value to display to the player. Given the car's current position, orientation, and speed on the race track, the RAD computes the trajectories that the car would follow if the player was to steer fully to the left or right. It models these trajectories as circular arcs which we denote as $\widehat{CL}$ and $\widehat{CR}$, respectively. The radii of the arcs are modeled as being directly proportional to the car's current speed, where the constant of proportionality represents how sharply the car turns. Through manual tuning, we found its value in our prototype game to be roughly 1.6.

Next, the RAD finds the points at which the trajectories intersect the track's edges, then it computes the respective arc lengths $l\widehat{CL}$ and $l\widehat{CR}$ from the car's position to these points. $l\widehat{CL}$ and $l\widehat{CR}$ represent the distances the car would travel before hitting an edge were the

player to steer fully to the left or right, respectively. Finally, the RAD sets the sound slider value to the following quantity, which we call the *time-to-impact ratio*:

$$\text{Slider Display Value} = \frac{l\widehat{CL}}{l\widehat{CL} + l\widehat{CR}}. \tag{6.1}$$

The sound slider's leftmost and rightmost positions are represented by zero and one respectively. The system will set the slider value to something different than the time-to-impact ratio in two cases. The first case is when both trajectories hit the track's left edge — which means that the player is driving toward the left edge — or when the player's car is currently off the track on the left side. The second is the analogous case for the track's right edge. In these cases, the system will set the slider value to zero and one, respectively.

From Lateral Position to Relative Risk

The algorithm described above represents a new approach for letting players know where they are situated on a racetrack. Unlike the stereo pan values in *Blindfold Racer*, *Mach 1*, and the *Top Speed* series, the sound slider's display value is *not* a direct reflection of the car's lateral position on the track. Rather, it is a function of the car's relative risk of hitting the track's left or right sides if the player wanted to. This distinction is what makes the sound slider intuitive even with the complex vehicle physics, steering behaviors, and track geometries that are present in modern racing games.

Figure 6.4 illustrates the benefit of updating the auditory display using our trajectory-based approach over the car's lateral position alone. The car's lateral position is the same between Figures 6.4(a) and (b) and between Figures 6.4(c) and (d), but the player's relative

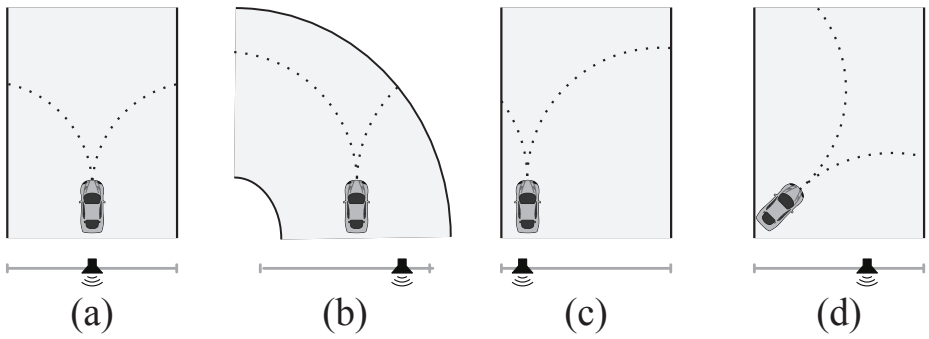


Figure 6.4: Four car poses and their corresponding sound slider values. Though the car's lateral position is the same between (a) and (b) and between (c) and (d), the corresponding sound slider values are very different. This is because the left and right trajectories' lengths — and therefore the relative risks of hitting the left and right sides of the track — are very different in each pair of cases.

risks of hitting the track's left and right sides is very different between each pair. In Figure 6.4(b), for example, the player is much more at risk of hitting the track's right side than they are in Figure 6.4(a) due to the sharp left turn in Figure 6.4(b), and the player should be aware of this.

As another example, the car's heading in Figure 6.4(c) puts the car more at risk of hitting the track's left edge than its right edge, while its heading in Figure 6.4(d) does the opposite. The player should be aware of this as well. The sound slider's trajectory-based approach communicates these risks.

Overcoming the Intention–Efficiency Tradeoff

We argue that the RAD's trajectory-based approach to computing its sound slider's displayed value allows it to overcome the intention–efficiency tradeoff that plagues other blind-accessible racing game interfaces (Figure 6.2).

The reason is that this approach distills many pieces of information — the car's lateral position on the track, its heading with respect to the track's, its speed, the track's width,

whether the track is about to immediately turn, and more — into a single measure that we hope is as relevant to the process of racing as all of that information put together. Moreover, it does so in a way that gives players the freedom to decide how risky they would like to race: whether they should cut corners by being close to hitting the track's inside edge or stay closer to the track's center. Sucu and Folmer's haptic driving interface, by comparison, eliminates intention by simply telling players which way they should steer at any given time.

We liken this process of distilling the many pieces of information to that of dimensionality reduction in machine learning and statistics. Dimensionality reduction is important in these fields because it boosts classification speed and removes redundancies in the representations of features. In the RAD's sound slider's case, the process reduces the amount of information that must be conveyed to the player while preserving its meaning and relevancy.

The RAD's Turn Indicator System

The RAD's turn indicator system uses spatialized (3D) sound cues to alert players of the direction, sharpness, and timing of upcoming turns and the length of in-progress turns. It works by playing a series of four beeps that trigger when the player's car crosses four corresponding and equally spaced distance markers placed ahead of the turn. The last beep is a continuous sound that begins playing just as the turn begins and continues sounding until the player completes the turn. Left and right turns are indicated by beeps emitted from the left and right ends of the sound slider's track, respectively. Overlapping turns are

indicated by overlapping sets of beeps.

By using four beeps to indicate turns, the player is given enough time to recognize the beeps' rhythm and anticipate the timing of the last beep, which marks the beginning of the turn. The player can then time their steering accordingly, cutting the corner if they wish by starting to steer a little before the last beep begins sounding. The distance markers triggering the four beeps are spaced 20 m apart, giving the player 1.7 s of advance notice of the turn when they are driving at the maximum speed of 35 m/s (approximately 75 mph).

The beep sounds themselves are modified recordings of a distant engine hum, adapted from [CosmicD 2007]. Low pitched beeps indicate soft turns, moderately pitched beeps indicate moderate turns, and high-pitched beeps indicate sharp turns. We defined soft turns as those which turn less than 0.3° per meter of track and sharp turns as those which turn more than 1° per meter of track. When a turn changes sharpness partway through, as in Turns 7a and 7b in Figure 6.6, the system treats each part as a separate turn and alerts the player accordingly.

In addition to playing the beeps, the system announces each upcoming turn's number, where Turn 1 is the track's first turn, Turn 2 the second, and so on. The number is announced at the same time as the first beep, and the goal is to help players learn the track over time as sighted players do.

Supported Actions

The RAD's sound slider and turn indicator system work together to support the following actions:

Understand the car's current speed: The sound slider's car engine sound will increase in pitch as the engine revs up, giving the player a general sense of the car's current speed.

Align the car with the track's heading: If the player's car is not aligned with the track's heading, the car engine sound will begin moving left or right on the sound slider. The player can align their car with the track's heading by steering until the engine sound stops moving.

Learn the track's layout: The turn indicator's turn number announcements help the player remember specific turns and sequences of turns.

Profile upcoming turns: The direction, sharpness, timing, and length of upcoming turns are indicated by the turn indicator beeps' left vs. right location in the soundscape, the beeps' pitch, the beeps' rhythm, and the fourth beep's duration, respectively.

Cut corners: By steering into a turn just before the turn indicator's fourth beep, the player can cut corners. The player can maintain an inside position during the turn by steering such that the engine sound moves toward the inside of the turn on the sound slider (and away from the slider's center).

Choose an early or late apex: By steering into a turn just before or after the turn indicator's fourth beep, the player can choose between taking an earlier apex or a later apex [Seas 2012].

Position the car for an optimal turning path: By steering the car in a way that moves the engine sound to a desired position on the sound slider ahead of a turn, the player can position the car for a more optimal driving path.

Know when braking is needed to complete a turn: The sound slider emits a tire

Figure 6.5: Racing game prototype implemented in Unity.

screeching sound when the player is going too fast in a turn to turn sharply enough.

6.3 Racing Game Prototype

As a proof of concept, we developed a racing game using the Unity game engine (version 5.4.2) [Unity Technologies 2016] and implemented the RAD in that racing game. Our prototype, shown in Figure 6.5, is an extension of TurnTheGameOn's Racing Game Template [O'Donnell 2015]. It features full 3D graphics and uses realistic vehicle physics from the Edy's Vehicle Physics package [García 2015].

Our game is played with a Sony DUALSHOCK 4 (PlayStation 4 controller) [Sony Interactive Entertainment 2013] and a standard pair of headphones. The controls are mapped similarly to other PlayStation 4 racing games: the left analog stick controls steering, R2 (the right analog trigger) is gas/acceleration, L2 (the left analog trigger) is brake and reverse, and R1 (the right shoulder button) is the handbrake. In case of a crash, participants

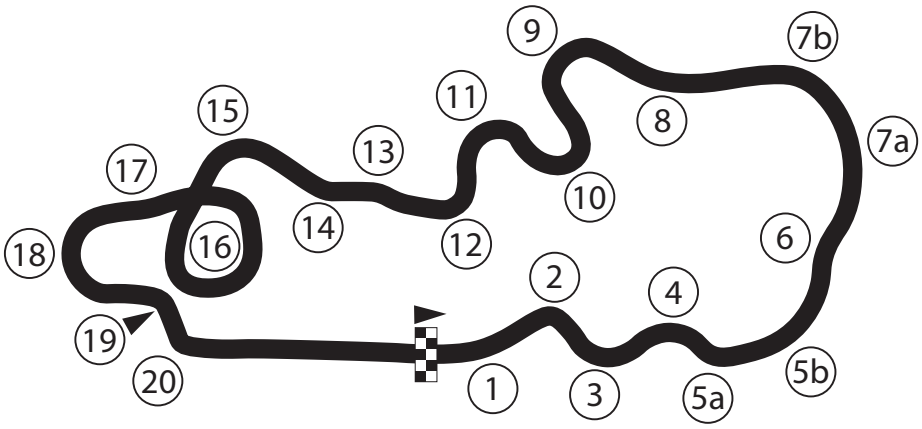


Figure 6.6: Circuit diagram for the racetrack used in our RAD user studies. This track is difficult and much more complex than ones in previous blind-accessible games, featuring a wide variety of turns.

could press the Triangle button to reset their car to the center of the track.

To generate spatialized (3D) sound, we enabled the simple demo spatializer provided by Unity's Audio Spatializer SDK [Unity Technologies 2017]. The spatializer applies a direct head-related transfer function (HRTF) that is based on a data set generated from a KEMAR dummy-head [Gardner and Martin 1995].

The Racetrack

Figure 6.6 shows the track that we used for our user studies. The track was developed internally at Unity [Unity Technologies 2015] and is much more complex than ones in previous blind-accessible games. It features a wide variety of turns: soft, moderate, and sharp turns; a long straightaway; a series of hairpin turns (Turns 9–11) that require players to slow down; a 270° turn (Turn 16); several short kinks in the track (Turns 8, 13, 14, & 17); several series of esses (Turns 1–5 & 19–20); long and gradual turns (Turns 5b & 7a); and turns that vary in sharpness as they progress (Turns 5, 7 & 17). The track is 3,641 m

long, 19 m wide, and has 20 turns in total.

6.4 Study 1: The RAD vs. Other Interfaces

We performed a study with both blind users and sighted users wearing blindfolds to compare the RAD with *Mach 1*'s interface [audiogames archive 2015] and Sucu and Folmer's haptic steering interface [Sucu and Folmer 2014]. These interfaces represent a broad range of design alternatives.

Our study had three goals. First, we wanted to determine how well the average person would perform with each of these user interfaces with a short amount of training. Second, we wanted to see how users would rank the three interfaces by order of preference. Third, we wanted to observe how well each interface helped players anticipate upcoming turns.

Study 1 Participants

Our study included fifteen participants. Three of them — P4, P8, and P11 — were blind their entire lives and the rest were sighted but blindfolded. Seven were age 16–25 and the rest were age 26–35; four were female and the rest were male. Our study was approved by our institution's Institutional Review Board, and parents were present with minors.

We recruited P4, P8, and P11 through Helen Keller Services for the Blind. P4 had no prior experience with racing games, while P8 and P11 had played just one audio racing game each years prior: *Top Speed* and *Blindfold Racer*, respectively. P8, however, described himself as a gamer and had played other types of audio games before, namely an RPG [Driftwood Audio Entertainment 2010] and a first-person shooter [Kaldobsky 2011].

Six of the twelve sighted participants had at least a moderate amount of experience playing video games, and the rest had very little experience. Of those with moderate experience, three would describe themselves as gamers.

We should note that participants who are sighted but blindfolded are generally not suitable proxies for participants who are blind. Silverman et al. [Silverman, Gwinn, and Van Boven 2015] found, for example, that sighted but blindfolded participants can be biased by the initial challenge of becoming blind, therefore judging the capability of people who are blind as much less than it actually is. As a result, and as is good practice [Sears and Hanson 2012], we will present the results from these two groups of participants separately.

Study 1 Procedure

In the study, participants raced using each of the three user interfaces in a counterbalanced order while we observed them. Participants controlled their car using a Sony DUALSHOCK 4 (PlayStation 4 controller) [Sony Interactive Entertainment 2013] and wore a pair of AmazonBasics on-ear headphones [Amazon.com, Inc. 2014]. All sighted participants wore blindfolds and could not see us loading the track, nor could they see what they were doing in the games. We told the participants that our team developed all three of the user interfaces. Each session lasted approximately two hours.

For both the haptic steering interface and the RAD, we had participants play our prototype racing game in which we implemented both. For *Mach 1*'s interface, however, we had participants play *Mach 1* itself. We did this because *Mach 1* uses simplified models rather than realistic designs for its tracks and steering system, and its user interface was designed

with the simplified models in mind. Since we loaded *Mach 1* into a level before the study began, the participants were not aware that they were playing a previously published game.

Like other modern game controllers, the DUALSHOCK 4's rumble motors are different in size, with the left motor being significantly larger than the right motor. Since the haptic steering interface requires identical rumble motors for the user's left and right hands, however, we replaced our DUALSHOCK 4's left motor with one identical to the right motor. We clamped the motors' vibration intensity to 50% of its normal maximum to make it easier for players to distinguish between the motors.

We began each user interface trial by training each participant with hands-on instruction for 15–20 minutes on how to use the interface. We created two training tracks in our prototype — a square track with rounded corners and a figure eight track — to help the participants relate the interfaces' feedback with easily understandable shapes. We told participants to play with each interface until they understood how they worked.

We followed the three trials with a survey asking participants to rank the three interfaces from their most to least favorite, rate how well each interface helped them anticipate upcoming turns on a 20-point Likert scale in which higher values were better, and offer feedback justifying their ratings. Participants' feedback was extensive. To analyze it, we first transcribed it in full, then — via a series of repeated readings — wrote topic labels for each piece describing what it was talking about. We then tallied positive and negative opinions for each identified topic. We report these numbers along with the quotes that were most descriptive and representative of overall opinions.

Study 1 Results: Participants Who Are Blind

User Interface Ranking

P4 ranked the user interfaces from best to worst as *Mach 1*'s interface, the RAD, and Sucu and Folmer's interface, in that order. Both P8 and P11 ranked them as the RAD, Sucu and Folmer's interface, and *Mach 1*'s interface, in that order.

Awareness of Upcoming Turns

On a 20-point Likert scale in which higher values are better, P4 rated their ability to anticipate upcoming turns using the RAD, Sucu and Folmer's interface, and *Mach 1*'s interface as 8, 11, and 15, respectively. P8's ratings were 18, 11, and 7, respectively, while P11's were 5, 10, and 12, respectively. The difference is sharp between P8 and the others. Both P4 and P11 had very little experience playing video games while P8 considers himself a gamer. Although P11 rated the RAD lowest and *Mach 1*'s interface highest on this scale, she ranked the RAD as the best of the three interfaces overall and *Mach 1*'s interface the worst of the three.

Driving Performance in Our Prototype Game

Of the participants who are blind, only P8 was able to complete a full lap, and he did so with each of the three user interfaces. P8, the only self-described gamer among the three, completed our track (Figure 6.6) with zero major collisions on his first try with both Sucu and Folmer's interface and the RAD.

Neither P4 nor P11 could complete a full lap using any of the user interfaces, though

all three participants completed our square and figure eight training tracks using both the RAD and Sucu and Folmer's interface. Recall that there were no training tracks for *Mach 1*. We should note that our track (Figure 6.6) resembles what one would find in a real video game and is very challenging compared to ones in existing blind-accessible racing games. Sucu and Folmer, for example, tested a basic oval and still found many crashes [Sucu and Folmer 2013; Sucu and Folmer 2014].

Qualitative Feedback: Mach 1's Interface

P4 rated *Mach 1*'s user interface as his favorite because it was the only one to explicitly read out the car's lateral position and because he felt that he "had [more] time to think and react" to its cues compared to the other interfaces. This is likely because *Mach 1* does not provide continuous feedback about the car's positioning as the other interfaces do; rather, it reads the information whenever a particular button is pressed.

Both P8 and P11 found *Mach 1*'s interface to be the worst of the three, with P11 saying that it was "the hardest" and "hard to use properly." P8 said that while "it had pretty much [all of the game elements that] [he] would expect from a racing game," it was "very difficult [to use because] there are so many things going on" at the same time, including many "sounds that are not relevant." He also said that it "was difficult [. . .] knowing when you are in the turn and when you are out of the turn" because the steadily increasing sound effect volume that it employs to indicate the beginnings of turns was not precise.

Qualitative Feedback: Haptic Steering Interface

P4 considered Sucu and Folmer's haptic steering interface to be the worst of the three "mainly due to not being able to see upcoming turns." P8 and P11 ranked the haptic steering interface in between their most and least favorite, with P11 saying that she "did not get to think about how to attack the turn[s]" and that "[using] it would have been easier if there was a warning in advance, when you should start turning." Still, P11 felt that while the lateral positioning feedback "wasn't exact[ly precise], it was to the point that I [...] could kind of tell if the car wasn't in the center."

P8 said that the vibrations "didn't give much [of an] indication of how sharp [each] turn was," preventing him from making strategies such as, "I shouldn't turn too much here to avoid colliding with the [inside] wall." He felt that "the experience would be better, perhaps, by "mak[ing] the game controller vibrate more or less" in intensity depending on the sharpness of the turn. Sucu and Folmer, however, found users' performance with continuous vibration feedback to be worse than with binary (on/off) feedback [Sucu and Folmer 2013; Sucu and Folmer 2014].

Qualitative Feedback: Racing Auditory Display (RAD)

P4 ranked the RAD in between his most and least favorite, saying "it is definitely better than the vibration method" (Sucu and Folmer's interface) but that he "still had a hard time" because it was "confusing to parse between the two types of sounds" (the sound slider and the turn indicator system). Both P8 and P11 considered the RAD to be their favorite interface, with P8 saying that it was "very, very logically built up [. . .] because it gave

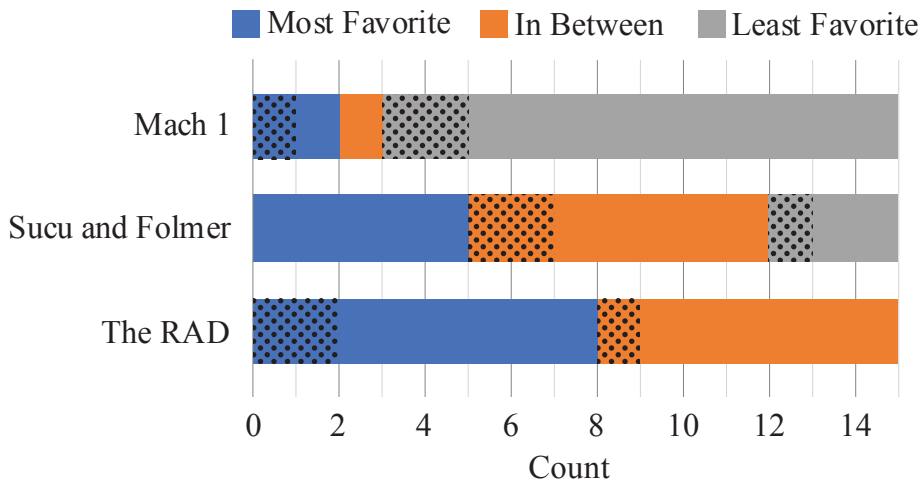


Figure 6.7: Participants' user interface rankings. The dot patterns indicate rankings from participants who are blind. Most disliked *Mach 1*'s interface, and eight of fifteen preferred the RAD's the most.

[him] an indication of how sharp the turns were [and] for how long [he was] in [each] turn."

P8 felt that distinguishing between soft, moderate, and sharp turns "worked very well with the tonality of the sound." P11, on the other hand, said that while she "got the concept, it was [. . .] harder to put the concept into use," finding the RAD "difficult to [learn] but very entertaining" to play with. She remarked that with the RAD "the feeling of the game is fast-paced," adding, "Yes, you have the time [to plan], but sometimes you might not be able to [pull it off]." P8 said that he liked how the RAD did not "constantly sa[y] 'Do this, do that,"' and followed up by saying, "After the training was done, I had the possibility of doing whatever I wanted to." These last two comments suggest that the RAD supports intention.

Study 1 Results: Sighted but Blindfolded Participants

User Interface Ranking

Figure 6.7 shows how participants ranked each interface from most to least favorite. Sighted participants' rankings are those without the dot patterns. Six sighted participants chose the RAD as their preferred interface, five chose the haptic steering interface, and one chose *Mach 1*'s interface. Ten out of twelve sighted participants liked *Mach 1*'s interface the least.

Awareness of Upcoming Turns

An ANOVA showed that the user interface has a significant main effect on the sighted participants' awareness of upcoming turns ($F_{2,22} = 4.83$, $p = 0.02$). Pairwise mean comparison showed that the only significant difference was between the RAD and *Mach 1*'s interface ($p < 0.05$). The mean (std. dev) ratings for this metric for the RAD, the haptic steering interface, and *Mach 1*'s interface are 13.0 (4.9), 8.8 (6.5), and 6.8 (5.5), respectively. This suggests that the RAD does a better job communicating the nature of upcoming turns for sighted players than *Mach 1*'s sound effects of increasing volume.

Driving Performance in Our Prototype Game

Ten out of twelve sighted participants were able to complete the track in Figure 6.6, five of which after crashing and resetting themselves many times. Their performance seemed to depend on their prior experience with video games: participants tended to perform well with both interfaces or poorly with both interfaces. All seven sighted participants with

at least moderate video game experience completed the track, two of whom after crashing many times. By contrast, only three of the five participants with limited video game experience completed the track, all of whom after crashing many times.

These results suggest that both the RAD and the haptic steering interface make it possible for gamers to play racing games without sight, but neither can make a non-gamer proficient at playing racing games.

Qualitative Feedback: Mach 1's Interface

Of the three interfaces, sighted participants liked *Mach 1*'s the least in general. Though many mentioned that "it was relatively easier to understand [their] horizontal location with [this interface's spoken] numeric value[s]" (P2) than with the other interfaces' feedback, four lamented that "having numbers read to [them] took extra brain power [to process, making] it much more difficult for [them] to move forward quickly" (P5). All said that it "took [them] a while to sort out all the sounds that were going on" (P15) and that there was "too much auditory information for too long a period" (P9).

Ten felt that determining the position and length of turns was "very difficult" (P2) and that they could not determine the turns' sharpness at all because "the sound leading up to the thump which indicates [when] turn[s begin and end were] more confusing and disorienting than anything" (P13). A different set of ten felt that a "[big] difficulty was to determine the difference between the probe number [(lateral position)] and the speed of the vehicle" (P10).

Qualitative Feedback: Haptic Steering Interface

Ten sighted participants felt that this interface's vibrations were "easier to [learn and] focus on [compared to] the [other interfaces'] multiple sounds" (P10), but five felt that "turning and preparing for turns was completely out of [their] control" (P5) because "the only interaction [they] had was immediately responding to the vibrations" (P5), "conforming to the rumble indicators" (P14), or as P13 put it, "just [. . .] bouncing around from wall to wall trying to stay in the center." P5 added that she "had no idea when a turn was coming up, how sharp or long it would be, [or] whether or not it was actually a turn [she] was dealing with or simply trying to straighten [her]self out on a straightaway after a turn."

Some liked how "the rumble [being] binary [made it] really clear [to know] when you are 'good' or 'bad'" (P13) but six bemoaned the resulting lack of intention (though not using that word). Three mentioned that they would prefer having differing levels of vibration so they could tell "exactly how far [. . .] from the middle of the road" (P6) they are or "how sharp the turn was" (P1, P3). As mentioned earlier, however, Sucu and Folmer found that users crashed much more with such a system than with binary vibration feedback [Sucu and Folmer 2013; Sucu and Folmer 2014]. We implemented a version of the user interface in which the vibration intensity varied based on how far the car is from the track's center line and found, via small-scale testing, this to be true as well.

Qualitative Feedback: Racing Auditory Display (RAD)

Eight sighted participants felt that the RAD's turn indicator system made them "well aware of the upcoming turns with their position and sharpness" (P3). Two of them, however, men-

tioned that the system was "sometimes confusing when [turns were] very short..." — in which case the fourth turn indicator beep would be very short — "...and/or followed immediately by another turn" (P1) — in which case the RAD would output multiple overlapping sets of beeps.

Four participants found the RAD difficult to use while two found it very natural. In particular, eight participants found it difficult to distinguish between the sound slider's engine sound and the turn indicators, with P5 mentioning that "as a full-sighted person [she is] not used to using every single sound as an informational cue and usually do[es]n't pay attention to such noises as engine volume." P2 and P5 sometimes found the RAD's sound slider "difficult to understand" (P2, P5) because "the location of the engine sound (left vs. right vs. middle) [can] change incredibly fast" as they enter sharp turns.

Seven participants mentioned that they were "almost always aware of which side of the track [they are] on" (P3) when using the RAD, with P3 adding, "[...] compared to both [of] the other methods where I was quite clueless." Seven participants felt that the RAD made them "fe[el] the most like [they were] racing" (P13) compared to the other interfaces. P9 found the RAD "fun and definitely the most immersive" of the three interfaces, and that with the RAD he "could actually visualize the car and its location."

6.5 Study 2: Field Test With Gamer Who Is Blind

Our second study tests whether the RAD makes it possible for a player who is blind to race better than Sucu and Folmer's haptic steering interface does, and whether their racing performance can match that of a sighted player's.

Study 2 Procedure

In this study, we had participant P8 from our first study — our only participant that is both blind and considers himself a gamer — drive thirteen laps around the racetrack in Figure 6.6 using Sucu and Folmer's haptic steering interface and fourteen laps using the RAD. We recorded his lap times, full driving paths, and gameplay video of him racing as he played. The car starts at the beginning of the long straightaway in Figure 6.6 so that it can reach full speed by the start of the first lap. Our supplemental video shows P8's third lap ever on this track.

We then had eight sighted players (three female, five male) drive one to three laps around the track using sight as we recorded their lap times and driving paths. We used just one to three laps here because we found in a pilot study that sighted players' lap times did not improve over the course of driving 14 laps. The same was true for P8: his average lap time for his first three laps was 0.3 s faster than for his last three.

Study 2 Results

Figure 6.8 compares lap times for the three conditions: P8 using the haptic steering interface, P8 using the RAD, and sighted players using vision. The mean (std. dev) lap times are 128.2 s (8.2 s), 117.0 s (3.7 s), and 111.7 s (3.5 s), respectively. An ANOVA showed that the user interface has a significant main effect on the mean lap times ($F_{2,32} = 23.38$, $p < 0.0001$). Pairwise mean comparison showed that the differences were significant between every pair of interfaces ($p < 0.01$) except the RAD vs. sighted players using vision. This suggests that the RAD allowed P8 to race significantly better than the haptic steering inter-

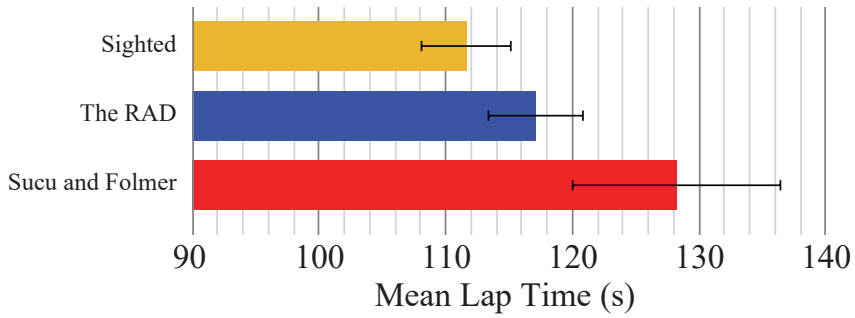


Figure 6.8: Mean lap times of P8 — a gamer who is blind — using Sucu and Folmer's haptic steering interface [Sucu and Folmer 2014], P8 using the RAD, and sighted players using vision. The error bars indicate standard deviations. With the RAD, P8 races significantly better than he does using the haptic steering interface and comparably to casual players racing with sight.

face did — saving an average of 11.2 s per lap — and comparably to that of players using sight. Only one of the sighted players, however, described themselves as a gamer.

Figure 6.9 compares typical driving paths from P8 using the haptic steering interface and the RAD, respectively. The haptic steering interface causes P8 to oscillate around the track's center line for the entire lap, which is this interface's usual behavior since it works by vibrating the player's controller when their heading is too far away from that of a center target point [Sucu and Folmer 2014]. By contrast, P8 drives in a much smoother path using the RAD. In Figure 6.9(b), for example, we see that P8 carves a nearly straight path through Turns 19 and 20 (which form an ess turn sequence) when using the RAD but follows the track's center line when using the haptic steering interface.

The mean (std. dev) driving path lengths are 3,639 m (74 m), 3,557 m (40 m), and 3,469 m (71 m) for the three respective conditions: P8 using the haptic steering interface, P8 using the RAD, and sighted players using vision. An ANOVA showed that the user interface has a significant main effect on the driving path length ($F_{2,32} = 19.21$, $p < 0.0001$). Pairwise mean comparison showed that the differences were significant between every pair

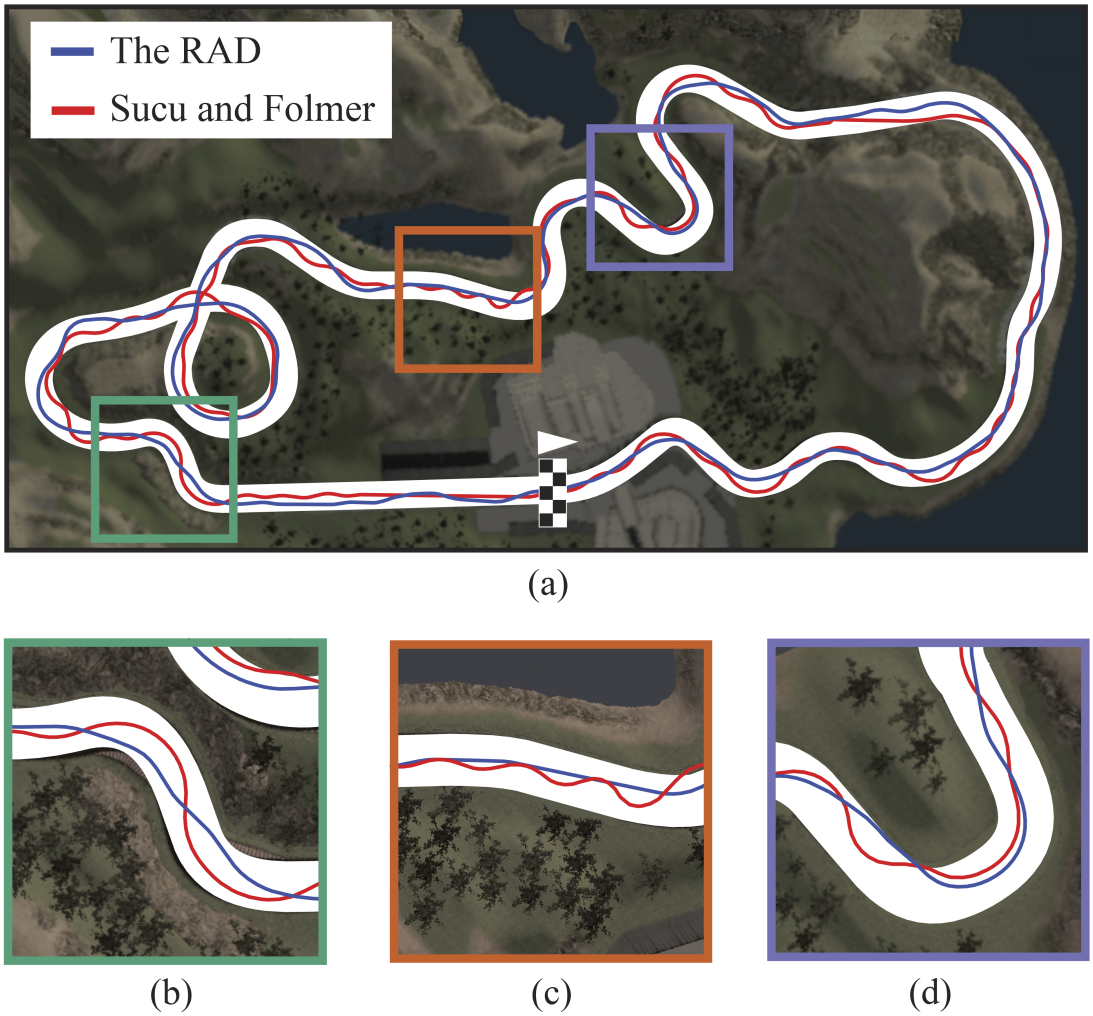


Figure 6.9: Sample driving paths of P8 — a gamer who is blind — using the RAD and using Sucu and Folmer's haptic steering interface [Sucu and Folmer 2014]. We compare the paths for (a) the entire circuit, (b) an ess turn, (c) a near-straight section, and (d) a hairpin turn. P8 oscillates constantly with the haptic steering interface but drives more smoothly when using the RAD. He is also able to cut the corners in (b) using the RAD. Our supplemental video shows P8's third lap with the RAD's audio included.

of interfaces ($p < 0.05$ for the haptic steering interface vs. the RAD and $p < 0.01$ otherwise). This shows that P8 can perform shorter laps with the RAD than with the haptic steering interface (mainly by reducing oscillations), though not quite as short as laps made by players driving with sight.

6.6 Human–Computer Interaction (HCI) Implications

Though games especially benefit from intention, our work has broader implications within HCI. First, our definition of a sound slider is generic: a virtual speaker that indicates a value within a range by its position on a 3D line segment in the soundscape. For blind users, sound sliders can substitute for traditional UI sliders; brightness, temperature, or pressure gauges; progress bars; and any other display that displays a value within a range. They can also help users perform steering tasks in the classical sense [Accot and Zhai 1997] by representing a tunnel's width.

Furthermore, the RAD can be used in place of AudioGPS [Holland, Morse, and Gedenryd 2002] and SWAN [Wilson, Walker, Lindsay, Cambias, and Dellaert 2007] for pedestrian navigation tasks. AudioGPS and SWAN tell users know which way to walk, but the RAD can tell users how wide the path or bridge is, how much "wiggle room" they have, and whether they are in the middle or toward one side, helping them avoid oncoming foot traffic.

6.7 Discussion

This chapter offers a vision of how video games can go beyond just being blind-accessible to being *equivalently accessible* to people who are blind, allowing them to play with a similar sense of control (intention) and efficiency as sighted players can. To this end, we introduce the *racing auditory display (RAD)* to help racing games become equivalently accessible to people who are blind. It comprises two novel sonification techniques: the *sound slider* for understanding a car's speed and trajectory on a racetrack and the *turn indicator system* for alerting players of the direction, sharpness, length, and timing of upcoming turns.

Through a pair of empirical studies, we found that players preferred the RAD's interface over that of *Mach 1*, a popular blind-accessible racing game, and at times "felt like [they] had as much information as if [they] could see the track" (P1). We demonstrated that the RAD makes it possible for a gamer who is blind to race comparably to casual players using sight.

In Section 7.2, we describe the limitations of this chapter's user studies, the limitations of the RAD itself, and what we believe to be promising future work for user interfaces such as the RAD.

Chapter 7

Conclusions, Limitations, and Future Work

7.1 Summary of Contributions

In this dissertation, we described the concept of *unmediated interaction* as an interaction modality that we should strive for when designing computing devices to reduce or eliminate the burden of using those devices. We identified two instances of that burden: the overhead that users must undergo to provide input to the device, which we called *input overhead*, and the overhead that users must undergo to interpret output from the device, which we called *output overhead*. We argued that by eliminating input and output overhead from our interaction with devices, we can make it seem like those devices are not even there and that we are accomplishing computing tasks using our own abilities or powers rather than intermediate devices.

We then, in the bulk of this dissertation, introduced three computational methods for reducing input overhead and one for reducing output overhead. The methods cover a broad range of domains and intersect several fields including machine learning, computer vision, optimization, acoustics, and game design.

First, in Chapter 3 we show how we can make it possible to eliminate the need for user inputs altogether via *input data mining* using *input words*. Namely, we show how prob-

abilistic topic models such as latent Dirichlet allocation and the player–gameplay action model, the latter of which we develop, can help us draw insights about video game players and the levels that they are playing — insights that can be used as a basis for recognizing players and personalizing their experience, all without their explicit input.

Next, in Chapter 4 we introduced *gaze locking*, a novel interaction modality for providing basic input in a nearly instantaneous way. Gaze locking is the notion of sensing eye contact directly from an image using a standard camera or existing images such as ones on the Web. By simplifying the continuous gaze tracking problem into the binary gaze locking problem, our gaze locking detector can exploit the special appearance of direct eye gaze, allowing devices to sense eye contact with over 90% accuracy at distances of up to 18 m. This in turn allows people to interact with computers, devices, and other objects just by looking at them.

In Chapter 5 we investigated how to make typing on small devices faster and less error-prone than it is when using the standard Qwerty keyboard. This work addresses instances in which users must provide devices with complex input and in which simply looking at the devices would not be enough to specify that input. Specifically, we explored how to modify Qwerty to make word gestures that are used for gesture typing shorter and more distinct, and how to do so in a way that prevents users from having to learn how to type all over again. By performing a rigorous optimization procedure using three metrics that we develop, we discovered keyboard layouts that are not too different from Qwerty and that can reduce error rates by 52% over Qwerty.

Last, in Chapter 6 we investigated the problem of reducing output overhead to make racing games accessible to people who are blind. We introduced the *racing auditory display*

(RAD), an audio system that works with a standard pair of headphones and that makes it possible for people who are blind to play the same types of racing games as sighted players can with a similar speed and sense of control to what sighted players have. The RAD works by using computation on the current game state to present players who are blind with stimuli that allows them to make the same moment-to-moment decisions that sighted players make while they race. We found that We also found that the RAD allows an avid gamer who is blind to race as well on a complex racetrack as casual sighted players can, without a significant difference between lap times or driving paths.

Together, we hope that these systems open the door to even more efforts in unmediated interaction, with the goal of making computers less like devices that we use and more like abilities or powers that we have.

7.2 Limitations and Future Work

I will close by offering my thoughts on each of my contributions' limitations, where I see each of my contributions potentially going next, and what I feel needs to happen to continue the path toward unmediated interaction.

Input Mining

Our methods for understanding and describing gameplay have several important limitations, however. First, our method is unlikely to produce meaningful results for genres in which the player does not have precise control of the game's character — genres such as point-and-click games and RPGs. We also do not show how to form input words from

analog input. Second, it is difficult to assign semantic meaning to some of the discovered gameplay types, and the number of output gameplay types K must be chosen *a priori*. Third, although Figures 3.7 and 3.14 can be used to recommend levels with gameplay that is similar to levels that players like, we do not evaluate players' perceptions of such recommendations. More importantly, the PGA model is but a first step towards removing factors that confound our understanding of the gameplay types present in a game. Our player recognition system's accuracy suggests that we controlled for the effect of a player's play style, but there are still other factors that affect the controller inputs entered by the player. One such factor is the current game state: in *Super Mario Bros. 3*, for example, pressing the B button rapidly makes Mario throw fireballs, but that is only possible if Mario touched a Fire Flower. Modeling event logs alongside controller inputs to include such factors is promising future work.

We should also be clear that when we say that our methods can be used to recommend stages to players based on ones that they liked before, we do not mean to suggest that there is no value in variety or that players will always prefer stages with similar gameplay over stages with different gameplay. Variety is important in games and the concept of game flow stems directly from this fact. Our methods are meant to serve as a new tool that game developers can use to verify that their game levels feature the desired type of gameplay without needing to hold formal playtests with live players.

Finally, our treatment of gameplay is limited to the types of action present in a game, and a full understanding of gameplay would only be possible if we also measured what players are feeling in response to the game's output. Such a move points back to Figure 3.1, in which every node and edge represents a signal that gives a unique perspective

of the experiences that players have with games. Future game analytics systems may analyze several or all of these signals in tandem to better understand and quantify what those experiences are.

Gaze Locking

There is great potential for future work in gaze locking using embedded cameras. Our sample detector consists of fairly simple mathematical operations, so future efforts could create a "gaze locker"—a camera module with a system-on-chip for gaze locking. This gaze locker would be small, cheap, and computationally efficient. Systems-on-chip already exist in cameras for applications such as exposure compensation and image compression.

Moreover, our gaze locking approach is passive and, as a result, energy-efficient. Hence, gaze lockers may even be able to employ energy-harvesting techniques like RFID tags do. We could also use gaze lockers to aid people who are blind and deaf by sensing when others are looking at them. Furthermore, if we place gaze lockers in many objects, they would collectively form a cloud that serves as a ubiquitous gaze tracker that is accurate over distance. For all of these reasons, we believe gaze lockers could be the perfect platform for bringing gaze-based interactive systems into everyday use in the future.

Gesture Typing

Although the nature, size, and complexity of multidimensional gesture typing optimization have surpassed its precedents in the literature [Bi, Smith, and Zhai 2010; Dunlop and Levine 2012; Lewis, Kennedy, and LaLomia 1999; Rick 2010], many questions beyond the

scope of this work require further research. These include further, larger, and longitudinal empirical studies of the multiple optimality dimensions. Further empirical investigation may redefine some or all of the optimality dimensions identified in this work in order to advance the gesture typing paradigm toward new shorthand writing systems that tolerate user errors, require minimal visual attention and motor effort, and remain easy to learn.

The RAD

Both our user studies and the RAD itself have several limitations that we would like to describe. First, our study included just four self-described gamers and three people who are blind, so our results cannot be assumed to apply to everyone from these groups. A more thorough follow-up study targeting gamers who are blind would be needed for this. Second, the RAD relies on 3D sound spatialization. Not everyone can hear spatialized sounds correctly with off-the-shelf head-related transfer functions (HRTFs). Future games could allow players to load an HRTF from a profile so they can hear spatialized sound clearly in many different games.

Last, the RAD is not as effective with non-gamers and does not teach them "video game literacy" such as how video game vehicle handling works, nor is it effective at helping players recover from crashes or from driving off the track. A future version of the RAD could include a *Mach 1*-style probing feature for helping players learn the game mechanics and recover from crashes. We also think it would be feasible to extend the RAD to incorporate other racing game elements such as opponent vehicles, boosts, item pickups, and shortcuts.

We hope that just as user interface toolkits provide tools such as scrollbars, sliders,

menus, and radio buttons that "just work" when software is published, game engines will one day include building blocks such as walls and track pieces that will "just work" with user interfaces such as the RAD or AudioGPS [Holland, Morse, and Gedenryd 2002] when games are published to make all games blind-friendly.

Privacy Implications

I would like to conclude with a note about privacy. The techniques that we are proposing for facilitating unmediated interaction, such as equipping devices with cameras for sensing eye contact and observing users' raw inputs as they use devices, have privacy implications that we would like to address. Hence, I am interested in developing imaging and/or vision approaches for sensing attention that can provably protect users' privacy, meaning for example that a gaze locking sensor can detect gaze locking and nothing more. A promising technique for doing so is to use a camera design with many fewer pixels than conventional cameras; it is possible, for example, to perform object tracking using just four pixels [Pooj, Grossberg, Belhumeur, and Nayar 2018], so it may be possible to sense attention using a small number as well.

Giving users privacy while continuously observing their raw inputs is a more difficult challenge. My belief is that most users do not object to having their usage tracked by a piece of software for the purpose of assisting the user within that software, but are instead wary of software exporting that information to a central server or a third party. Hence, my interest is in keeping usage information local — on a user's person — and that information could physically follow them throughout the day as they use different devices.

Towards Unmediated Interaction

I have so far divided efforts in facilitating unmediated interaction into those that have aim to reduce input overhead and those that aim to reduce output overhead, and I feel that doing so is useful for describing future efforts in this space as well. Moreover, the three different approaches that I have identified for reducing input overhead — namely, eliminating the need for user inputs altogether, making the process of providing basic inputs near-instantaneous, and making the process of typing complex inputs less laborious — reveal some ideas that I feel are particularly promising.

Take, for example, the approach of eliminating the need for user inputs altogether. In this dissertation, we focused only on input mining for doing so, and just a particular brand of input mining as well, but of course there are many other techniques that we can use for doing so. One exciting idea is to incorporate a brain–computer interface to help predict what users may be thinking of doing or acting upon next. Even a very rough prediction can help cull a number of unlikely interactions considerably. Another idea is to develop methods for devices to share information with each other to help each other predict what users will do next. Text mining and crowdsourcing researchers have already developed models for predicting which fine-grained actions are likely to follow from others that have been observed [Fast, McGrath, Rajpurkar, and Bernstein 2016], so the tools for acting upon this type of information already exist.

Regarding the goal of making basic inputs nearly instantaneous, there are many new input modalities that we can create to go beyond gaze locking. One of the limitations of gaze locking, for example, is that it can only facilitate binary interactions: either the user

is looking at the camera or they are not. Gaze-based interfaces would be much more useful if they were capable of recognizing other canonical commands such as “Up,” “Down,” “Left,” “Right,” and “Back.” Hence, a future technique may combine eye contact with a basic gesture as part of a universal grammar for interacting with devices.

New interaction techniques could also be used to make typing less laborious for cases in which complex input must be specified. For example, Vulcan is a word gesture keyboard that works with users’ fingers in mid-air [Markussen, Jakobsen, and Hornbæk 2014] — a new technique could take this a step further by allowing users to control faraway devices by gesturing words on the palms or the backs of their hands.

Bibliography

Accot, Johnny and Shumin Zhai (1997). "Beyond Fitts' Law: Models for Trajectory-based HCI Tasks." In: *Proc. ACM SIGCHI Conf. Hum. Fact. in Comput. Sys. (CHI 1997)*. New York, NY, USA: ACM Press, pp. 295–302. DOI: http://dx.doi.org/10.1145/258549.258760.

Allman, Troy, Rupinder K. Dhillon, Molly A.E. Landau, and Sri H. Kurniawan (2009). "Rock Vibe: Rock Band® Computer Games for People with No or Limited Vision." In: *Proceeding Elev. Int. ACM SIGACCESS Conf. Comput. Access. (ASSETS 2009)*. New York, New York, USA: ACM Press, pp. 51–58. DOI: http://dx.doi.org/10.1145/1639642.1639653.

Amazon.com, Inc. (2014). *AmazonBasics Lightweight On-Ear Headphones*. Retrieved September 17, 2017 from http://a.co/9OFo5NC.

Android Open Source Project (2018). *Android Open Source Project*. Retrieved August 16, 2018 from https://source.android.com.

Argyle, M. and J. Dean (1965). "Eye-contact, distance and affiliation." In: *J. Sociometry* 28.3, pp. 289–304.

Atkinson, Matthew Tylee and Sabahattin Gucukoglu (2004). *The AGRIP Project (Audio-Quake)*. Retrieved September 16, 2017 from http://agrip.org.uk.

Baluja, Shumeet and Dean Pomerleau (1994). *Non-Intrusive Gaze Tracking Using Artificial Neural Networks*. Tech. rep. Department of Computer Science, Carnegie Mellon University.

Beymer, D. and M. Flickner (2003). "Eye Gaze Tracking Using an Active Stereo Head." In: *Proc. CVPR 2003*. IEEE Press. DOI: 10.1109/CVPR.2003.1211502.

Bi, Xiaojun, Shiri Azenkot, Kurt Partridge, and Shumin Zhai (2013a). "Octopus: evaluating touchscreen keyboard correction and recognition algorithms via remulation." In: *Proc. CHI 2013*. ACM Press, pp. 543–552. ISBN: 9781450318990. DOI: 10.1145/2470654.2470732. URL: http://dl.acm.org/citation.cfm?id=2470654.2470732.

Bi, Xiaojun, Shiri Azenkot, Kurt Partridge, and Shumin Zhai (2013b). "Octopus: evaluating touchscreen keyboard correction and recognition algorithms via "remulation"." In: *Proc. CHI 2013*. ACM Press, pp. 543–552.

Bi, Xiaojun, Ciprian Chelba, Tom Ouyang, Kurt Partridge, and Shumin Zhai (2012). "Bimanual gesture keyboard." In: *Proc. UIST 2012*. ACM Press, pp. 137–146.

Bi, Xiaojun, Tom Ouyang, and Shumin Zhai (2014). "Both complete and correct? Multi-objective optimization of a touchscreen keyboard." In: *Proc. CHI 2014*. ACM Press, pp. 2297–2306.

Bi, Xiaojun, Barton A. Smith, and Shumin Zhai (2010). "Quasi-Qwerty soft keyboard optimization." In: *Proc. CHI 2010*. ACM Press, pp. 283–286.

— (2012). "Multilingual touchscreen keyboard design and optimization." In: *Human–Computer Interaction* 27.4, pp. 352–382.

Bíró, István, Dávid Siklósi, Jácint Szabó, and András A. Benczúr (2009). "Linked latent Dirichlet allocation in web spam filtering." In: *Proc. AIRWeb 2009*. ACM Press, pp. 37–40. ISBN: 9781605584386. DOI: 10.1145/1531914.1531922. URL: http://dl.acm.org/citation.cfm?id=1531914.1531922.

Blackwell, H. Richard (1946). "Contrast thresholds of the human eye." In: *J. Opt. Soc. Am.* 36.11. DOI: 10.1364/JOSA.36.000624. URL: http://www.opticsinfobase.org/abstract.cfm?URI=josa-36-11-624.

Blei, D. M. and M. I. Jordan (2003). "Modeling annotated data." In: *Proc. SIGIR 2003*. ACM Press, pp. 127–134. ISBN: 1581136463. DOI: 10.1145/860435.860460. URL: http://dl.acm.org/citation.cfm?id=860435.860460.

Blei, David M. (2012). "Probabilistic topic models." In: *Commun. ACM* 55.4, pp. 77–84. ISSN: 00010782. DOI: 10.1145/2133806.2133826. URL: http://dl.acm.org/ft{_}gateway.cfm?id=2133826{\&}type=html.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan (2003). "Latent dirichlet allocation." In: *J. Mach. Learn. Res.* 3, pp. 993–1022. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944919.944937.

Bolt, Richard A. (1980). "Put-that-there: Voice and gesture at the graphics interface." In: *Proc. SIGGRAPH 1980*. ACM Press, pp. 262–270. ISBN: 0-89791-021-4. DOI: 10.1145/800250.807503. URL: http://doi.acm.org/10.1145/800250.807503.

Buckley, David, Ke Chen, and Joshua Knowles (2013). "Predicting skill from gameplay input to a first-person shooter." In: *Proc. CIG 2013*. IEEE Press, pp. 105–112. ISBN: 978-

1-4673-5311-3. DOI: 10.1109/CIG.2013.6633655. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6633655.

Cao, Xiang and Shumin Zhai (2007). "Modeling human performance of pen stroke gestures." In: *Proc. CHI 2007*. ACM Press, pp. 1495–1504.

Chang, C.C. and C.J. Lin (2011). "LIBSVM: a library for support vector machines." In: *ACM Trans. Intell. Syst. Technol.* 2.3, 27:1–27:27.

Chen, Milton (2002). "Leveraging the asymmetric sensitivity of eye contact for videoconferencing." In: *Proc. CHI 2002*. ACM Press, pp. 49–56.

Chien, Jen-Tzung and Chuang-Hua Chueh (2008). "Latent dirichlet language model for speech recognition." In: *Proc. SLT 2004*. IEEE Press, pp. 201–204. ISBN: 978-1-4244-3471-8. DOI: 10.1109/SLT.2008.4777875. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4777875.

Church, Doug (1999b). *Formal Abstract Design Tools*. Gamasutra. Retrieved August 25, 2017 from http://www.gamasutra.com/view/feature/131764/formal_abstract_design_tools.php.

— (1999a). "Formal Abstract Design Tools." In: *Game Developer Magazine* 6.7, p. 28.

— (2006). "The Game Design Reader: A Rules of Play Anthology." In: Cambridge, MA, USA: MIT Press. Chap. Formal Abstract Design Tools, pp. 366–381.

Cline, Marvin G. (1967). "The Perception of Where a Person Is Looking." In: *Am. J. Psychol.* 80.1, pp. 41–50.

CosmicD (2007). *engine_hum_new.wav*. Audio file. Retrieved September 14, 2017 from https://freesound.org/s/33503/.

Crawford, Chris (2012). "Verb thinking." In: *Chris Crawford on Interactive Storytelling*. 2nd ed. New Riders, pp. 81–92.

Curcio, Christine A., Kenneth R. Sloan, Robert E. Kalina, and Anita E. Hendrickson (1990). "Human photoreceptor topography." In: *J. Comp. Neurol.* 292.4, pp. 497–523. ISSN: 1096-9861. DOI: 10.1002/cne.902920402. URL: http://dx.doi.org/10.1002/cne.902920402.

Drachen, Anders, Alessandro Canossa, and Georgios N. Yannakakis (2009). "Player modeling using self-organization in Tomb Raider: Underworld." In: *Proc. CIG 2009*. IEEE Press, pp. 1–8. ISBN: 978-1-4244-4814-2. DOI: 10.1109/CIG.2009.5286500. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5286500.

Drachen, Anders, Rafet Sifa, Christian Bauckhage, and Christian Thurau (2012). "Guns, swords and data: clustering of player behavior in computer games in the wild." In: *Proc. CIG 2012*. IEEE Press, pp. 163–170. ISBN: 978-1-4673-1194-6. DOI: 10.1109/CIG.2012.6374152. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6374152.

Driftwood Audio Entertainment (2010). *Entombed - An RPG Roguelike Game for the Blind and Visually Impaired*. Retrieved January 2, 2018 from http://www.blind-games.com/entombed.aspx.

Duda, R.O., P.E. Hart, and D.G. Stork (2001). "Pattern classification." In: vol. 2. Wiley-Interscience, pp. 114–124.

Dunlop, Mark and John Levine (2012). "Multidimensional pareto optimization of touch-screen keyboards for speed, familiarity and improved spell checking." In: *Proc. CHI 2012*. ACM Press, pp. 2669–2678. DOI: 10.1145/2207676.2208659.

El-Nasr, Magy Seif, Anders Drachen, and Alessandro Canossa (2013). *Game Analytics: Maximizing the Value of Player Data*. Springer, p. 815. ISBN: 1447147693. URL: https://books.google.com/books?hl=en{\&}lr={\&}id=-guHadEPrFcC{\&}pgis=1.

EyeTech (2013). *EyeTech™ VT2 XL*. http://www.eyetechds.com/vt2-xl.shtml.

Fast, Ethan, William McGrath, Pranav Rajpurkar, and Michael S. Bernstein (2016). "Augur: Mining Human Behaviors from Fiction to Power Interactive Systems." In: *Proc. CHI 2016*. ACM Press, pp. 237–247.

GMA Games (2005). *Shades of Doom Version 1.2*. Retrieved September 16, 2017 from http://www.gmagames.com/sod.html.

Gamer, M. and H. Hecht (2007). "Are you looking at me? Measuring the cone of gaze." In: *J. Exp. Psychol. [Hum Percept.]* 33.3, pp. 705–715.

Garber, Megan. *You Probably Write a Novel's Worth of Email Every Year*. The Atlantic (Jan. 8, 2013). http://www.theatlantic.com/technology/archive/2013/01/you-probably-write-a-novels-worth-of-email-every-year/266942/. Accessed August 19, 2018.

García, Angel "Edy" (2015). *Edy's Vehicle Physics*. Retrieved September 15, 2017 from http://www.edy.es/dev/vehicle-physics/.

Gardner, William G. and Keith D. Martin (1995). "HRTF Measurements of a KEMAR." In: *J. Acoust. Soc. Am.* 97.6, pp. 3907–3908. DOI: http://dx.doi.org/10.1121/1.412407.

Gemmell, J., K. Toyama, C.L. Zitnick, T. Kang, and S. Seitz (2000). "Gaze awareness for video-conferencing: a software approach." In: *IEEE Multimedia* 7.4, pp. 26–35.

Getschow, Christian O., Michael J. Rosen, and Cheryl Goodenough-Trepagnier (1986). "A systematic approach to design a minimum distance alphabetical keyboard." In: *Proc. RESNA 1986*. RESNA, pp. 396–398.

Gibson, James J. and Anne D. Pick (1963). "Perception of Another Person's Looking Behavior." In: *Am. J. Psychol.* 76.3, pp. 386–394.

Gow, Jeremy, Robin Baumgarten, Paul Cairns, Simon Colton, and Paul Miller (2012). "Unsupervised modeling of player style with LDA." In: *IEEE Trans. Comput. Intell. AI Games* 4.3, pp. 152–166. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2012.2213600. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6269992.

Hansen, D.W. and Q. Ji (2010). "In the eye of the beholder: A survey of models for eyes and gaze." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 32.3, pp. 478–500.

Hansen, Dan Witzner and Arthur E. C. Pece (2005). "Eye tracking in the wild." In: *Comput. Vis. Image Und.* 98.1, pp. 155–181. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2004.07.013. URL: http://dx.doi.org/10.1016/j.cviu.2004.07.013.

Harrison, Chris (2013). "The Human Body as an Interactive Computing Platform." PhD thesis. Carnegie Mellon University.

Hastings, Erin J. and Kenneth O. Stanley (2010). "Galactic arms race." In: *ACM SIGEVOlution* 4.4, pp. 2–10. ISSN: 19318499. DOI: 10.1145/1810136.1810137. URL: http://dl.acm.org/citation.cfm?id=1810136.1810137.

Hastings, W. Keith (1970). "Monte Carlo sampling methods using Markov chains and their applications." In: *Biometrika*. Vol. 57. 1, pp. 97–109.

Hedgpeth, Terri, John A Black Jr., and Sethuraman Panchanathan (2006). "A Demonstration of the iCARE Portable Reader." In: *Proc. 8th Int. ACM SIGACCESS Conf. Comput. Access. (ASSETS 2006)*. New York, New York, USA: ACM Press, pp. 279–280. ISBN: 1-59593-290-9. DOI: 10.1145/1168987.1169054. URL: http://portal.acm.org/citation.cfm?doid=1168987.1169054http://doi.acm.org/10.1145/1168987.1169054.

Holland, Simon, David R. Morse, and Henrik Gedenryd (2002). "AudioGPS: Spatial audio navigation with a minimal attention interface." In: *Pers. Ubiquitous Comput.* 6.4, pp. 253–259. ISSN: 16174909. DOI: 10.1007/s007790200025. URL: http://link.springer.com/10.1007/s007790200025.

Homan, Matthew D. and Andrew Gelman (2014). "The no-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2638586.

Hunicke, R., M. LeBlanc, and R Zubek (2004). "MDA: a formal approach to game design and game research." In: *Proc. AAAI Work. Chal. Game AI*. Vol. 4. AAAI Press, pp. 1–5.

Hurst, Amy, Scott E. Hudson, and Jennifer Mankoff (2007). "Dynamic detection of novice vs. skilled use without a task model." In: *Proc. CHI 2007*. ACM Press, p. 271. ISBN: 9781595935939. DOI: 10.1145/1240624.1240669. URL: http://dl.acm.org/citation.cfm?id=1240624.1240669.

Hutchins, Edwin L., James D. Hollan, and Donald A. Norman (1985). "Direct manipulation interfaces." In: *Human–Computer Interaction* 1.4, pp. 311–338.

Itsuki, Hiroshi, Asuka Takeuchi, Atsushi Fujita, and Hitoshi Matsubara (2010). "Exploiting MMORPG log data toward efficient RMT player detection." In: *Proc. ACE 2010*. ACM Press, pp. 118–119. ISBN: 9781605588636. DOI: 10.1145/1971630.1971670. URL: http://dl.acm.org/citation.cfm?id=1971630.1971670.

Kaldobsky, Jeremy "Aprone" (2011). *Aprone's Accessible Software and Games*. Retrieved January 2, 2018 from http://www.kaldobsky.com/audiogames/.

Kang, Ah Reum, Jiyoung Woo, Juyong Park, and Huy Kang Kim (2013). "Online game bot detection based on party-play log analysis." In: *Comput. Math. with Appl.* 65.9, pp. 1384–1395. ISSN: 08981221. DOI: 10.1016/j.camwa.2012.01.034. URL: http://www.sciencedirect.com/science/article/pii/S0898122112000442.

Kendon, A. (1967). "Some functions of gaze direction in social interaction." In: *Acta Psychol.* 26, pp. 22–63.

Kim, Joy and Jonathan Ricaurte (2011). "TapBeats: Accessible and Mobile Casual Gaming." In: *Proc. 13th Int. ACM SIGACCESS Conf. Comput. Access. (ASSETS 2011)*. New York, New York, USA: ACM Press, pp. 285–286. DOI: http://dx.doi.org/10.1145/2049536.2049609.

Kim, Jun H., Daniel V. Gunn, Eric Schuh, Bruce Phillips, Randy J. Pagulayan, and Dennis Wixon (2008). "Tracking real-time user experience (TRUE)." In: *Proc. CHI 2008*. ACM Press, pp. 443–451. ISBN: 9781605580111. DOI: 10.1145/1357054.1357126. URL: http://dl.acm.org/citation.cfm?id=1357054.1357126.

Kim, Samuel, Shrikanth Narayanan, and Shiva Sundaram (2009). "Acoustic topic model for audio information retrieval." In: *Proc. WASPAA 2009*. IEEE Press, pp. 37–40.

ISBN: 978-1-4244-3678-1. DOI: 10.1109/ASPAA.2009.5346483. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5346483.

Kiss, Jemima (2012). *Facebook hits 1 billion users a month*. http://www.guardian.co.uk/technology/2012/oct/04/facebook-hits-billion-users-a-month.

Kristensson, Per Ola and Shumin Zhai (2004). "SHARK^2: a large vocabulary shorthand writing system for pen-based computers." In: *Proc. UIST 2004*. ACM Press, pp. 43–52.

LG USA (2018). *LG Nexus 5: Made For What Matters for Sprint in Black*. Retrieved August 16, 2018 from https://www.lg.com/us/cell-phones/lg-D820-Sprint-Black-nexus-5.

Laurel, Brenda K. (1986). "User Centred system design: new perspectives on human-computer interaction." In: ed. by Donald A. Norman and Stephen W. Draper. Morgan Kaufmann. Chap. Interface as mimesis, pp. 139–158.

Lewis, James R., Peter J. Kennedy, and Mary J. LaLomia (1999). "Development of a digram-based typing key layout for single finger/stylus input." In: *Proc. Hum. Fact. Ergon. Soc. 1999*. Sage Publications, pp. 415–419.

Lukins, Stacy K., Nicholas A. Kraft, and Letha H. Etzkorn (2008). "Source code retrieval for bug localization using latent Dirichlet allocation." In: *Proc. WCRE 2008*. IEEE Press, pp. 155–164. ISBN: 978-0-7695-3429-9. DOI: 10.1109/WCRE.2008.33. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4656405.

MacKensie, I. Scott and Shawn X. Zhang (1999). "The design and evaluation of a high-performance soft keyboard." In: *Proc. CHI 1999*. ACM Press, pp. 25–31.

MacKenzie, I. Scott, Shawn X. Zhang, and R. William Soukoreff (1999). "Text entry using soft keyboards." In: *Behaviour & Information Technology* 18.4, pp. 235–244. DOI: 10.1080/014492999118995.

Mahlmann, Tobias, Anders Drachen, Julian Togelius, Alessandro Canossa, and Georgios N. Yannakakis (2010). "Predicting player behavior in Tomb Raider: Underworld." In: *Proc. CIG 2010*. IEEE Press, pp. 178–185. ISBN: 978-1-4244-6295-7. DOI: 10.1109/ITW.2010.5593355. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5593355.

Mankoff, Jennifer and Gregory D. Abowd (1998). "Cirrin: a word-level unistroke keyboard for pen input." In: *Proc. UIST 1998*. ACM Press, pp. 213–214. DOI: 10.1145/288392.288611.

Markussen, Anders, Mikkel Rønne Jakobsen, and Kasper Hornbæk (2014). "Vulture: A Mid-air Word-gesture Keyboard." In: *Proc. CHI 2014*. ACM Press, pp. 1073–1082.

Martin, W. Wade and Robert F. Jones (1982). "The accuracy of eye-gaze judgement: A signal detection approach." In: *Brit. J. Soc. Psychol.* 21.4, pp. 293–299.

Maskeri, Girish, Santonu Sarkar, and Kenneth Heafield (2008). "Mining business topics in source code using latent Dirichlet allocation." In: *Proc. ISEC 2008*. ACM Press, pp. 113–120. ISBN: 9781595939173. DOI: 10.1145/1342211.1342234. URL: http://dl.acm.org/citation.cfm?id=1342211.1342234.

Matthews, B.W. (1975). "Comparison of the predicted and observed secondary structure of T4 phage lysozyme." In: *BBA-Protein Struct."* 405.2, pp. 442–451.

McMurrough, Christopher D., Vangelis Metsis, Jonathan Rich, and Fillia Makedon (2012). "An eye tracking dataset for point of gaze detection." In: *Proc. ETRA 2012*. ACM Press, pp. 305–308. ISBN: 978-1-4503-1221-9. DOI: 10.1145/2168556.2168622. URL: http://doi.acm.org/10.1145/2168556.2168622.

Miller, Daniel, Aaron Parecki, and Sarah A. Douglas (2007). "Finger Dance: A Sound Game for Blind People." In: *Proc. 9th Int. ACM SIGACCESS Conf. Comput. Access. (ASSETS 2007)*. New York, New York, USA: ACM Press, pp. 253–254. DOI: http://dx.doi.org/10.1145/1296843.1296898.

Mimno, David, Hanna M. Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum (2011). "Optimizing semantic coherence in topic models." In: *Proc. EMNLP 2011*. ACL Press, pp. 262–272. ISBN: 978-1-937284-11-4. URL: http://dl.acm.org/citation.cfm?id=2145432.2145462.

Moore, Edward F (1956). "Gedanken-experiments on sequential machines." In: *Automata studies* 34, pp. 129–153.

Morelli, Tony, John Foley, and Eelke Folmer (2010). "VI-Bowling: A Tactile Spatial Exergame for Individuals with Visual Impairments." In: *Proc. 12th Int. ACM SIGACCESS Conf. Comput. Access. (ASSETS 2010)*. New York, New York, USA: ACM Press, pp. 179–186. DOI: http://dx.doi.org/10.1145/1878803.1878836.

Morimoto, C.H., A. Amir, and M. Flickner (2002). "Detecting Eye Position and Gaze from a Single Camera and 2 Light Sources." In: *Proc. ICPR 2002*. IEEE Press, pp. 314–317. DOI: 10.1109/ICPR.2002.1047459.

Morimoto, C.H., D. Koons, A. Amir, and M. Flickner (2000). "Pupil detection and tracking using multiple light sources." In: *J. Image Vision Comput.* 18.4.

Morimoto, C.H. and M.R.M. Mimica (2005). “Eye gaze tracking techniques for interactive applications.” In: *Comput. Vis. Image Und.* 98.1, pp. 4–24.

National Federation of the Blind (2013). *Blind Driver Challenge*. Retrieved September 12, 2017 from `http://www.blinddriverchallenge.org`.

Nintendo (1988). *Super Mario Bros. 3*. Nintendo (1988). Video game.

Nishino, K. and S.K. Nayar (2004). “The world in an eye.” In: *Proc. CVPR 2004*. IEEE Press, pp. 444–451.

Norman, Don (2013). *The Design of Everyday Things: Revised and Expanded Edition*. Constellation. Chap. The Psychology of Everyday Actions, pp. 37–73.

O’Donnell, Stephen "TurnTheGameOn" (2015). *Racing Game Template*. Retrieved January 3, 2018 from `https://www.turnthegameon.com/racing-game-template`.

Omron (2012). *Omron. OKAO vision*. `http://www.omron.com/r_d/coretech/vision/okao.html`.

Oulasvirta, Antti, Anna Reichel, Wenbin Li, Yan Zhang, Myroslav Bachynskyi, Keith Vertanen, and Per Ola Kristensson (2013). “Improving Two-thumb Text Entry on Touchscreen Devices.” In: *Proc. CHI 2013*. ACM Press, pp. 2765–2774. DOI: `10.1145/2470654.2481383`.

Pedersen, Chris, Julian Togelius, and Georgios N. Yannakakis (2009). “Modeling player experience in Super Mario Bros.” In: *Proc. CIG 2009*. IEEE Press, pp. 132–139. ISBN: 978-1-4244-4814-2. DOI: `10.1109/CIG.2009.5286482`. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5286482`.

Perlin, Ken (1998). “Quikwriting: continuous stylus-based text entry.” In: *Proc. UIST 1998*. ACM Press, pp. 215–216.

Ponz, Victoria, Arantxa Villanueva, and Rafael Cabeza (2012). “Dataset for the evaluation of eye detector for gaze estimation.” In: *Proc. UbiComp 2012*. ACM Press, pp. 681–684. ISBN: 978-1-4503-1224-0. DOI: `10.1145/2370216.2370364`. URL: `http://doi.acm.org/10.1145/2370216.2370364`.

Pooj, Parita, Michael Grossberg, Peter Belhumeur, and Shree Nayar (2018). “The Minimalist Camera.” In: *Proc. BVMC 2018*. British Machine Vision Association.

Pritchard, Jonathan K., Matthew Stephens, and Peter Donnelly (2000). “Inference of population structure using multilocus genotype data.” In: *Genetics* 155.2, pp. 945–959. URL: `http://www.genetics.org/content/155/2/945.short`.

Rick, Jochen (2010). “Performance optimizations of virtual keyboards for stroke-based text entry on a touch-based tabletop.” In: *Proc. UIST 2010*. ACM Press, pp. 77–86.

Ruijter, Leonard de, Pieter de Ruijter, Bram Duvigneau, and Davy Loots (2004). *Playing in the Dark: Top Speed*. Retrieved September 16, 2017 from http://www.playinginthedark.net/topspeed_e.php.

Rutkowski, Chris (1982). “An Introduction to the Human Applications Standard Computer Interface Part 1: Theory and Principles.” In: *Byte* 7.10, pp. 291–310.

Sears, Andrew and Vicki L. Hanson (2012). “Representing users in accessibility research.” In: *ACM Trans. Access. Comput.* 4.2, pp. 1–6. DOI: http://dx.doi.org/10.1145/2141943.2141945.

Seas (2012). *Cornering Technique*. Retrieved September 14, 2017 from http://www.formula1-dictionary.net/cornering_tech.html.

Shaker, Noor, Georgios N. Yannakakis, and Julian Togelius (2011). “Feature analysis for modeling game content quality.” In: *Proc. CIG 2011*. IEEE Press, pp. 126–133. ISBN: 978-1-4577-0010-1. DOI: 10.1109/CIG.2011.6031998. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6031998.

Shechtman, D., P. Riordan-Eva, and P. Hardigan (2005). “Maximum angle of ocular duction during visual fixation as a function of age.” In: *Strabismus* 13.1, pp. 21–26.

Shell, Jeffrey S., Roel Vertegaal, Daniel Cheng, Alexander W. Skaburskis, Changuk Sohn, A. James Stewart, Omar Aoudeh, and Connor Dickie (2004). “ECSGlasses and EyePliances: using attention to open sociable windows of interaction.” In: *Proc. ETRA 2004*. ACM Press, pp. 93–100. ISBN: 1-58113-825-3. DOI: 10.1145/968363.968384. URL: http://doi.acm.org/10.1145/968363.968384.

Shneiderman, Ben (1983). “Direct manipulation: A step beyond programming languages.” In: *Computer* 8, pp. 57–69.

Shultz, Marty (2013). *Blindfold Racer: Creating the Track*. Blog post. Retrieved September 12, 2017 from https://blindfoldgames.org/2013/12/03/creating-the-track/.

— (2014b). *Blindfold Racer: It’s Too Hard to Control*. Blog post. Retrieved September 12, 2017 from https://blindfoldgames.org/2014/03/10/blindfold-racer-its-too-hard-to-control/.

— (2014a). *Blindfold Racer*. Retrieved September 16, 2017 from http://www.blindfoldracer.com.

Shultz, Marty (2015). *Blindfold Color Crush*. Retrieved September 16, 2017 from https://blindfoldgames.org/user-guides/blindfold-color-crush-user-guide/.

Silverman, Arielle M., Jason D. Gwinn, and Leaf Van Boven (2015). "Stumbling in Their Shoes: Disability Simulations Reduce Judged Capabilities of Disabled People." In: *Soc. Psychol. Personal. Sci.* 6.4, pp. 464–471. DOI: http://dx.doi.org/10.1177/1948550614559650.

Sivic, Josef, Bryan C. Russell, Andrew Zisserman, William T. Freeman, and Alexei A. Efros (2008). "Unsupervised discovery of visual object class hierarchies." In: *Proc. CVPR 2008*. IEEE Press, pp. 1–8. ISBN: 978-1-4244-2242-5. DOI: 10.1109/CVPR.2008.4587622. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4587622.

Smith, John D., Roel Vertegaal, and Changuk Sohn (2005). "ViewPointer: lightweight calibration-free eye tracking for ubiquitous handsfree deixis." In: *Proc. UIST 2005*. ACM Press, pp. 53–61. ISBN: 1-59593-271-2. DOI: 10.1145/1095034.1095043. URL: http://doi.acm.org/10.1145/1095034.1095043.

Sony Interactive Entertainment (2013). *DUALSHOCK 4 Wireless Controller*. Retrieved September 15, 2017 from https://www.playstation.com/en-in/explore/accessories/dualshock-4-wireless-controller/.

Steinfeld, Edward and Jordana Maisel (2012). *Universal Design: Creating Inclusive Environments*. Hoboken, NJ, USA: Wiley.

Stiefelhagen, Rainer, Jie Yang, and Alex Waibel (1996). "A model-based gaze tracking system." In: *Proc. IEEE Intelligence and Systems 1996*. IEEE Press, pp. 304–310.

— (1997). "Tracking eyes and monitoring eye gaze." In: *Proc. PUI 1997*.

Sucu, Burkay and Eelke Folmer (2013). "Haptic Interface for Non-Visual Steering." In: *Proc. 2013 Int. Conf. Intell. User Interfaces (IUI 2013)*. New York, New York, USA: ACM Press, pp. 427–434. DOI: http://dx.doi.org/10.1145/2449396.2449451.

— (2014). "The Blind Driver Challenge: Steering using Haptic Cues." In: *Proc. 16th Int. ACM SIGACCESS Conf. Comput. Access. (ASSETS 2014)*. New York, New York, USA: ACM Press, pp. 3–10. DOI: http://dx.doi.org/10.1145/2661334.2661357.

Symons, Lawrence A, Kang Lee, Caroline C Cedrone, and Mayu Nishimura (2004). "What are you looking at? Acuity for triadic eye gaze." In: *J. Gen. Psychol.* 131.4, pp. 451–469.

TASVideos community (2016). *Bizhawk*. TASVideos (2016). http://tasvideos.org/BizHawk.html. Accessed April 12, 2016. URL: http://tasvideos.org/BizHawk.html.

Tan, Kar-Han, David J. Kriegman, and Narendra Ahuja (2002). "Appearance-based Eye Gaze Estimation." In: *Proc. WACV 2002*. IEEE Press, pp. 191–195.

The Radicati Group, Inc. *Email Statistics Report, 2018-2022*. Retrieved August 19, 2018 from https://www.radicati.com/?p=15185.

Tran, T V, T Letowski, and K S Abouchacra (2000). "Evaluation of acoustic beacon characteristics for navigation tasks." In: *Ergonomics* 43.6, pp. 807–827. ISSN: 0014-0139. DOI: 10.1080/001401300404760. URL: http://www.ncbi.nlm.nih.gov/pubmed/10902889.

Turk, M.A. and A.P. Pentland (1991). "Face recognition using eigenfaces." In: *Proc. CVPR 1991*. IEEE Press, pp. 586–591.

Unity Technologies (2015). *Car Tutorial (Unity 3.x only)*. Retrieved September 16, 2017 from http://u3d.as/1qU.

— (2016). *Unity Download Archive*. Retrieved September 14, 2017 from https://unity3d.com/get-unity/download/archive.

— (2017). *Audio Spatializer SDK*. Retrieved September 15, 2017 from https://docs.unity3d.com/2017.2/Documentation/Manual/AudioSpatializerSDK.html.

Vertegaal, R. and J.S. Shell (2008). "Attentive user interfaces: the surveillance and sousveillance of gaze-aware objects." In: *Soc. Sci. Inf.* 47.3, pp. 275–298.

Vertegaal, Roel (2003). "Attentive user interfaces." In: *Comm. ACM* 46.3, pp. 31–33. ISSN: 0001-0782. DOI: 10.1145/636772.636794. URL: http://doi.acm.org/10.1145/636772.636794.

Vertegaal, Roel, Robert Slagter, Gerrit van der Veer, and Anton Nijholt (2001). "Eye gaze patterns in conversations: there is more to conversational agents than meets the eyes." In: *Proc. CHI 2001*. ACM Press, pp. 301–308. ISBN: 1-58113-327-8. DOI: 10.1145/365024.365119. URL: http://doi.acm.org/10.1145/365024.365119.

WBDG Accessible Committee, Jordana L. Maisel, and Molly Ranahan (2017). *Beyond Accessibility to Universal Design*. Retrieved August 22, 2017 from https://www.wbdg.org/design-objectives/accessible/beyond-accessibility-universal-design/.

Walker, Bruce N. and Jeff. Lindsay (2004). "Auditory navigation performance is affected by waypoint capture radius." In: *Proc. Int. Conf. Audit. Disp. (ICAD 2004)*. Sydney, NSW, AU: Georgia Inst. Tech., pp. 6–9. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.97.9443.

Walker, Bruce N. and Jeffrey Lindsay (2006). "Navigation Performance With a Virtual Auditory Display: Effects of Beacon Sound, Capture Radius, and Practice." In: *Hum. Factors* 48.2, pp. 265–278. DOI: http://dx.doi.org/10.1518/001872006777724507.

WeFightForever (2015). *Does anyone actually like water levels?* Reddit (2015). https://redd.it/3pvns5. Accessed April 6, 2016. URL: https://redd.it/3pvns5.

Weidenbacher, U., G. Layher, P.-M. Strauss, and H. Neumann (2007). "A comprehensive head pose and gaze database." In: *Proc. IE 2007*. IET Press. DOI: 10.1049/cp:20070407. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4449972&isnumber=4449897.

Westin, T (2004). "Game accessibility case study: Terraformers - A real-time 3d graphic game." In: *Proc. Fifth Int. Conf. Disabil. Virtual Real. Assoc. Technol. (ICDVRAT 2004)*. Reading, UK: University of Reading, pp. 95–100. DOI: 10.1.1.103.8041.

Wilson, Jeff, Bruce N. Walker, Jeffrey Lindsay, Craig Cambias, and Frank Dellaert (2007). "SWAN: System for wearable audio navigation." In: *Proc. IEEE Int. Symp. Wearable Comput. (ISWC 2007)*. Boston, MA, USA: IEEE Press, pp. 91–98. ISBN: 9781424414536. DOI: 10.1109/ISWC.2007.4373786. URL: http://ieeexplore.ieee.org/document/4373786/.

Xing, Dongshan and Mark Girolami (2007). "Employing latent Dirichlet allocation for fraud detection in telecommunications." In: *Pattern Recognit. Lett.* 28.13, pp. 1727–1734. ISSN: 01678655. DOI: 10.1016/j.patrec.2007.04.015. URL: http://www.sciencedirect.com/science/article/pii/S016786550700147X.

Yamada, Hisao (1980). "A historical study of typewriters and typing methods: from the position of planning Japanese parallels." In: *Information Processing* 2.4, pp. 175–202.

Yannakakis, G. N. and J. Togelius (2011). "Experience-driven procedural content generation." In: *IEEE Trans. Affect. Comput.* 2.3, pp. 147–161. ISSN: 1949-3045. DOI: 10.1109/T-AFFC.2011.6. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5740836.

Yannakakis, G.N. and J. Hallam (2009). "Real-time game adaptation for optimizing player satisfaction." In: *IEEE Trans. Comput. Intell. AI Games* 1.2, pp. 121–133. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2009.2024533. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5067382.

Ye, Zhefan, Yin Li, Alireza Fathi, Yi Han, Agata Rozga, Gregory D. Abowd, and James M. Rehg (2012). "Detecting eye contact using wearable eye-tracking glasses." In: *Proc. UbiComp 2012*. ACM Press, pp. 699–704.

Yuan, Bei and Eelke Folmer (2008). "Blind Hero: Enabling Guitar Hero for the Visually Impaired." In: *Proc. 10th Int. ACM SIGACCESS Conf. Comput. Access. (ASSETS 2008)*. New York, New York, USA: ACM Press, pp. 169–176. ISBN: 9781595939760. DOI: http://dx.doi.org/10.1145/1414471.1414503.

Yuan, Bei, Eelke Folmer, and Frederick C. Harris (2011). "Game accessibility: A survey." In: *Univers. Access Inf. Soc.* 10.1, pp. 81–100. DOI: http://dx.doi.org/10.1007/s10209-010-0189-5.

Zhai, Shumin, Michael Hunter, and Barton A. Smith (2000). "The Metropolis keyboard — an exploration of quantitative techniques for virtual keyboard design." In: *Proc. UIST 2000*. ACM Press, pp. 119–128.

— (2002). "Performance optimization of virtual keyboards." In: *Human–Computer Interaction* 17.2, pp. 229–269.

Zhai, Shumin and Per Ola Kristensson (2003). "Shorthand writing on stylus keyboard." In: *Proc. CHI 2003*. ACM Press, pp. 97–104.

— (2008). "Interlaced QWERTY — accommodating ease of visual search and input flexibility in shape writing." In: *Proc. CHI 2008*. ACM Press, pp. 593–596.

— (2010). "Text Entry Systems: Mobility, Accessibility, Universality." In: Morgan Kaufmann. Chap. Introduction to shape writing, pp. 139–158.

Zhai, Shumin, Carlos Morimoto, and Steven Ihde (1999). "Manual and gaze input cascaded (MAGIC) pointing." In: *Proc. CHI 1999*. ACM Press, pp. 246–253. ISBN: 0-201-48559-1. DOI: 10.1145/302979.303053. URL: http://doi.acm.org/10.1145/302979.303053.

audible-edge (2009). *Chrysler LHS tire squeal 04 (04-25-2009).wav*. Audio file. Retrieved September 14, 2017 from https://freesound.org/s/71739/.

audiogames archive (2015). *Games by Jim Kitchen*. Retrieved September 16, 2017 from http://www.agarchive.net/pages/devs/kitchensinc.html.

eyebox2 (2007). *eyebox2™ Impressions™ for Signage*. https://www.xuuk.com/Images/eyebox2productspecs.pdf.